TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra softwarového inženýrství

Studijní program: M2612 – Elektrotechnika a informatika
Studijní obor: 3902T005 – Automatické řízení a inženýrská informatika

Akční člen s CAN protokolem

An actuator with CAN protocol realization

Diplomová práce

Autor **Jakub Horák**

Vedoucí diplomové práce: Ing. Josef Grosman

V Liberci 4.5.2006

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č.121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašují, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užití mé diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum: 4.5.2006

Podpis:

Poděkování

Na tomto místě bych rád poděkoval všem lidem, kteří nějakým způsobem přispěli ke vzniku této diplomové práce.

Zvláštní poděkování patří vedoucímu diplomové práce, Ing. Josefu Grosmanovi, za konzultace a odbornou pomoc**.**

Dále bych chtěl poděkovat rodičům za poskytnuté zázemí a morální podporu.

Anotace

Cílem diplomová práce je navrhnout a realizovat uzel sběrnice CAN pro řízení akčního členu, kterým je krokový motorek. Uzel je vybaven řadičem CAN firmy Philips SJA1000 a budičem této sběrnice Philips PCA82C250. Řídícím mikropočítačem je Atmel AT90S8535 a obvodem pro řízení krokových motorků je SAA1027. Konfigurace modulu CAN je možná z PC po sériové lince RS232 a je uložena ve vnitřní paměti EEPROM. Program pro mikropočítač je napsán v jazyce C v prostředí CodeVisionAVR a program pro PC v prostředí Delphi 6.0. Pro ověření takto realizovaného uzlu je využit vývojový systém s mikropočítačem Infineon C167CR s integrovaným řadičem sběrnice CAN, pro nějž je napsán program ve vývojovém prostředí µVision3 od firmy Keil Electronics.

Abstract

The aim of the diploma thesis is to develope and to realize a CAN node for controlling of the actuator, which is a stepper motor. The node is equiped with CAN controller Philips SJA1000 and CAN driver Philips PCA82C250. The controlling microprocessor is Atmel AT90S8535 and the circuit for driving of stepper motors is Philips SAA1027. The node's configuration is able from PC via a serial interface RS232, and is loaded into the internal memory EEPROM of the microprocessor. The program for the microprocessor is written in language C in the development system CodeVisionAVR and the program for PC is written in environment Delphi 6.0. For the verification of this node's realization is used a development system with microcomputer Infineon C167CR, which has integrated CAN controller. The program is written in µVision3 environment from the Keil Electronics.

Klíčová slova

– sběrnice, protokol, mikroprocesor, řadič, motor

Obsah

Úvod

Se zaváděním dokonalejších a složitějších systémů vyžadujících přenos stále většího množství informací, se kladou čím dál tím vyšší nároky na komunikační cesty. Roste počet řídících jednotek s jednočipovými mikropočítači vyhodnocujícími veliké množství informací z čidel, či jiných řídících jednotek, pro jejich zpracování a následné ovládání různých akčních členů. Vyžaduje to mnoho vodičů vzájemně propojujících jednotlivé systémy mezi sebou. Z tohoto důvodu se zavádějí sběrnice, které pomáhají jejich počty snížit a usnadnit tím realizaci složitějších systémů. Jejich zavedením se sníží cena a hmotnost daného výrobku, která vyplývá ze snížení počtu vodičů a zjednodušení konstrukce svorkovnic, čímž se zároveň zvýší i spolehlivost přenášených dat. Zavádění sběrnic sebou samozřejmě nese i nějaké nevýhody. Těmi jsou např. zvýšení nároků na řídící jednotky, které v systému bez sběrnice nemusely být, což má za následek i tomu úměrné navýšení ceny. Dále také složitější odhalování některých závad, které je ovšem možné určitým způsobem kompenzovat zavedením kvalitní diagnostiky.

Mezi širokou škálou různých průmyslových komunikačních sběrnic si své místo velice silně vybudovala také sběrnice CAN (Controller Area Network). Na rozdíl od ostatních typů, které byly od počátku vyvíjeny jako prostředky pro komunikaci mezi automatizačními zařízeními na systémové úrovni, byla sběrnice CAN původně zamýšlena především pro použití v automobilech. Postupem času si také získala v tomto odvětví své dominantní postavení, a v současné době je vlastně standardem pro automobilové sběrnice. Zároveň ovšem začala být široce používána i jako průmyslová komunikační sběrnice na nižší systémové úrovni i na úrovni snímačů a akčních členů.

Tato diplomová práce se zabývá konkrétním návrhem a realizací uzlu pro připojení krokového motorku na sběrnici CAN. Nejprve jsou vysvětleny základní principy protokolu CAN a uvedeny technické prostředky pro jeho realizaci. V další části je již popsána vlastní hardwarová realizace takového uzlu s vybraným jednočipovým mikropočítačem, řadičem a budičem sběrnice CAN a obvodem pro ovládání krokových motorků. Konfigurace tohoto uzlu je možná prostřednictvím PC po sériové lince RS232, k čemuž slouží vytvořená aplikace ve vývojovém prostředí Delphi. Ověření činnosti je provedeno pomocí vývojového systému s mikropočítačem pro vytváření sběrnice CAN. Práce je zakončena závěrem, který obsahuje vyhodnocení návrhu a jeho možná vylepšení.

1 Popis CAN sběrnice

1.1 Historie a vlastnosti sběrnice CAN

Sběrnice CAN (*Controller Area Network*) byla vyvinuta v polovině osmdesátých let dvacátého století firmou Robert Bosch ve spolupráci s firmou Intel pro použití v automobilovém průmyslu. Původně byla zamýšlena především pro propojení elektronických zařízení uvnitř nákladních automobilů s cílem snížit komplikovanost a cenu stávajícího propojení provedeného klasickou kabeláží. Podobně se pracovalo po celém světě na dalších automobilových sběrnicích (např. VAN či ABUS), avšak CAN získal postupem času výsadní postavení mezi sběrnicemi určenými pro nasazení v automobilech. I když byla sběrnice CAN navržena pro automobilový průmysl, začala se již od počátku pro své dobré vlastnosti prosazovat i v průmyslových aplikacích v komunikaci řídících systémů i na úrovni snímačů a akčních členů. Příčinou je zejména nízká cena, snadné nasazení, vysoká přenosová rychlost, snadná rozšiřitelnost a dostupnost velkého množství elektronických komponentů a vývojových nástrojů. CAN je na své fyzické a spojové vrstvě definován mezinárodní normou ISO 11898. Pro standardizaci vyšších vrstev komunikačního protokolu vzniklo sdružení CiA (*CAN in Automation*), které v současné době podporuje následující varianty – CAL (CAN Aplication Layer)/CANopen, DeviceNet, CANKingdom a Smart Distributed System. Standard vyšších vrstev protokolu pro automobilové aplikace vznikl později po průmyslových standardech a byl nazván OSEK.

CAN byl navržen tak, aby umožnil provádět distribuované řízení systémů v reálném čase s přenosovou rychlostí do 1Mbit/s a vysokým stupněm zabezpečení přenosu proti chybám. Jedná se o protokol typu multi-master, kde všechny připojené uzly mají právo přístupu ke sběrnici, což přináší zjednodušení řízení a zvýšení spolehlivosti. Pro řízení přístupu k médiu je použita sběrnice s náhodným přístupem řešící kolize na základě prioritního rozhodování (CSMA/CD+AMP). Zprávy vysílané po sběrnici protokolem CAN neobsahují žádnou informaci o cílovém uzlu, a jsou přijímány všemi ostatními uzly. Každá zpráva obsahuje identifikátor udávající význam přenášené zprávy a její prioritu. Je zajištěno přednostní přijetí zprávy s vyšší prioritou a umožněno na základě identifikátoru zajistit, aby uzel přijímal zprávy pouze jemu určené.

1.2 Struktura vrstev sběrnice CAN podle normy ISO 11898

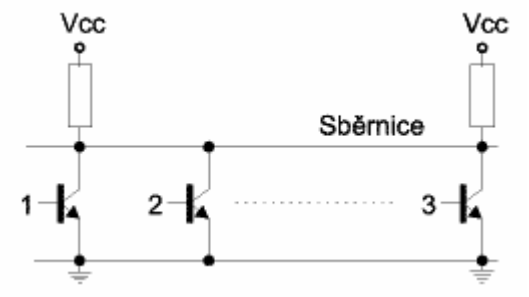
Protokol sběrnice CAN je podle normy ISO 11898 definován pouze na dvou nejnižších vrstvách referenčního modelu ISO/OSI, tj. na vrstvě fyzické a linkové. Na rovinně aplikační vrstvy (viz. kapitola 3.4) existuje několik nekompatibilních protokolů, které jsou sdruženy pod uživatelskou organizací CiA, o které bylo zmíněno již dříve.

Vrstva 7	Aplikační vrstva	
Vrstva 6	Prezentační vrstva	
Vrstva 5	Relační vrstva	
Vrstva 4	Transportní vrstva	
Vrstva 3	Síťová vrstva	
Vrstva 2	Spojová (linková) vrstva	
	Logické linkové řízení	- vysílání dat
	MAC	- vysílání požadavku na data - filtrování dat
	Řízení přístupu k médiu	- rámce a arbitráž
	LLC	- kontrola chyb - časování bitu
Vrstva 1	Fyzická vrstva	

Tab.1 Struktura vrstev CAN sběrnice podle referenční modelu ISO/OSI

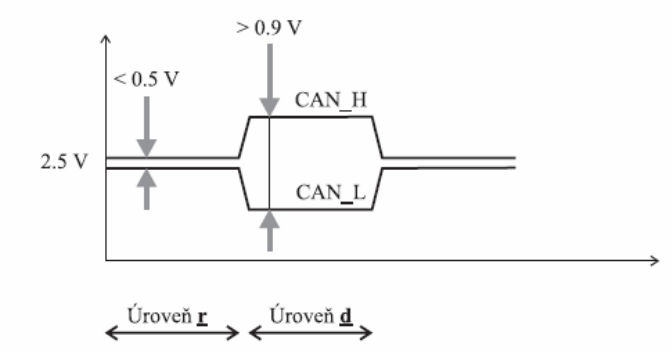
1.2.1 Fyzická vrstva

Základním požadavkem na fyzické přenosové médium protokolu CAN je, aby bylo možno realizovat funkci logického součinu. Jsou definovány logické úrovně dominantní (log.0) a recesivní (log.1) takovým způsobem, že pokud jsou současně vysílány recesivní bity, je i výsledný stav na sběrnici recesivní, je-li současně vysílán dominantní a recesivní bit, pak výsledný stav na sběrnici je dominantní. Příkladem takové realizace může být optické vlákno, kdy dominantnímu stavu bude odpovídat stav "svítí" a recesivnímu "nesvítí". Dalším příkladem může být sběrnice buzená hradly s otevřeným kolektorem (obr.1), kdy dominantnímu stavu odpovídá sepnutý tranzistor a recesivnímu rozepnutý.



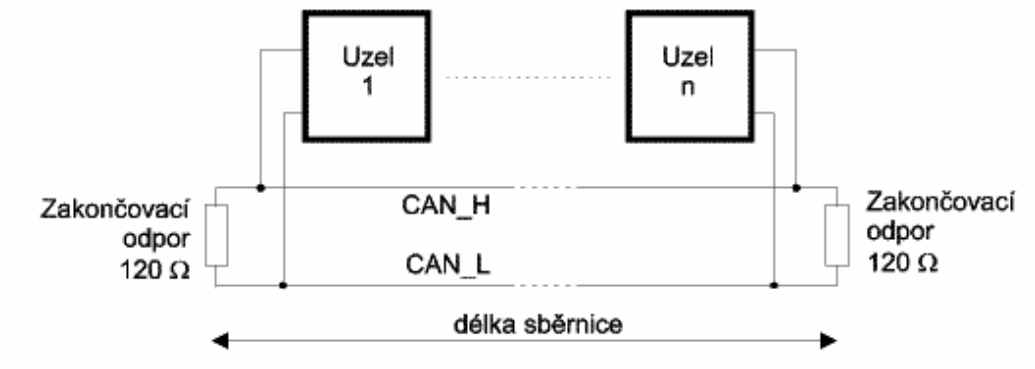
Obr.1 Příklad realizace fyzické vrstvy protokolu CAN viz.[12]

Pro realizaci přenosového média se nejčastěji používá diferenciální sběrnice dle normy ISO 11898. Tato norma definuje elektrické vlastnosti budičů, principy časování, synchronizace a kódovaní jednotlivých bitů. Sběrnice je tvořena dvěma vodiči CAN_H a CAN_L, kde recesivní úroveň je definována nulovým rozdílovým napětím (je vyhodnoceno i když U_{diff}<0,5V), a dominantní úroveň je definována rozdílovým napětím větším než U_{diff}>0,9V. Nominální hodnoty pro dominantní úroveň jsou U_{CAN_H}=3,5V a U_{CAN_l}=1,5V, a tedy U_{diff}=2V.



Obr.2 Reprezentace recesivní a dominantní úrovně viz [12]

Sběrnice je na obou koncích zakončena odpory 120Ω pro eliminaci odrazů na vedení. Připojení jednotlivých zařízení na sběrnice je nejčastěji provedeno pomocí konektorů D-SUB.



Obr.3 Principiální struktura sítě CAN podle ISO 11898 viz.[12]

Na sběrnici může být připojeno teoreticky neomezeně mnoho uzlů, avšak vzhledem k zatížení a zajištění správných statických a dynamických parametrů sběrnice je v normě uvedeno maximum 30 připojených uzlů na sběrnici. V normě je dále uvedena maximální délka vedení 40m pro přenosovou rychlost 1Mbit/s. Pro jiné přenosové rychlosti není délka sběrnice normou přesně definována, ale lze usoudit, že s klesající přenosovou rychlostí se bude možná maximální délka vedení zvyšovat, a to až na možných 10km při 5kbit/s.

1.2.2 Spojová vrstva

Spojová (linková) vrstva protokolu CAN je tvořena dvěma podvrstvami, MAC a LLC (viz. Tab.1). První z nich má na starosti řízení přístupu k médiu MAC (*Medium Access Control*) a jejím úkolem je starat se o kódování dat, časování bitů s možností vkládat doplňkové bity (*bit stuffing*), rozlišení priorit zpráv, detekce chyb a jejich hlášení a potvrzení o správnosti přijatých dat. Druhá podvrstva spojové vrstvy, logické linkové řízení LLC (*Logical Link Control*) má za úkol vysílání dat případně žádostí o data, filtrovat přijatá data a posílat hlášení o přetížení.

Jelikož řízení přístupu jednotlivého uzlu k médiu je náhodné, a tudíž může uzel začít vysílat kdykoli chce, je nutné nějakým způsobem vyřešit kolizi, která nastane v případě, že začnou vysílat uzly současně. Toto se řeší tzv. nedestruktivní arbitráží s prioritou danou identifikátory CSMA/CD+AMP (*Carrier Sense Multiple Access with Collision Detection and Arbitration on Message Priority*), tzn. nedojde k porušení vysílané zprávy s nejvyšší prioritou (nejnižší identifikátor). Identifikátor je umístěn na začátku vysílané zprávy. Každý uzel porovnává hodnotu vysílaného bitu se stavem

sběrnice, neshodují-li se (sběrnice je ve stavu dominant a uzel vysílá recesivní bit), pak okamžitě ukončí vysílání, počká až se sběrnice uvolní a pokusí se o opětovné vysílání.

Vzhledem k vysokým nárokům na zabezpečení přenášených dat, jsou zde implementovány některé další mechanizmy. Vedle již zmíněného sledování stavu sběrnice (*bit monitoring*) je to vkládání bitů (*bit stuffing*), CRC kód, kontrola formátu zprávy a potvrzování přijetí zprávy (*Acknowlege*). Vkládání bitů znamená, že po vyslání pěti bitů stejné úrovně je vložen bit opačné úrovně (*stuffing*) a na straně přijímače je opět odstraněn (*destuffing*). Toto je prováděno zejména z důvodu synchronizace, jelikož jsou přenášené zprávy poměrně dlouhé, ale také k detekování chyb. Dalším mocným nástrojem pro zabezpečení přenášených dat je doplnění zprávy o 15-ti bitový CRC kód (*Cyclic Redundancy Code*), který je generován ze všech předcházejících bitů příslušné zprávy podle polynomu: $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$. Posledním opatřením je kontrola zprávy (*Message Frame Check*), kdy jsou sledovány bity, které mají předepsanou hodnotu, v případě neshody je generována chyba formátu zprávy.

1.2.3 Základní typy zpráv

Protokol CAN definuje celkem čtyři typy zpráv, rozdělené na zprávy týkající se přenosu dat, mezi něž patří datové zprávy *(Data Frame*), zprávy žádosti o data (*Remote Frame*) a zprávy řídící komunikaci po sběrnici, jako chybové zprávy (*Error Frame*) a zprávy o přetížení (*Overload Frame*).

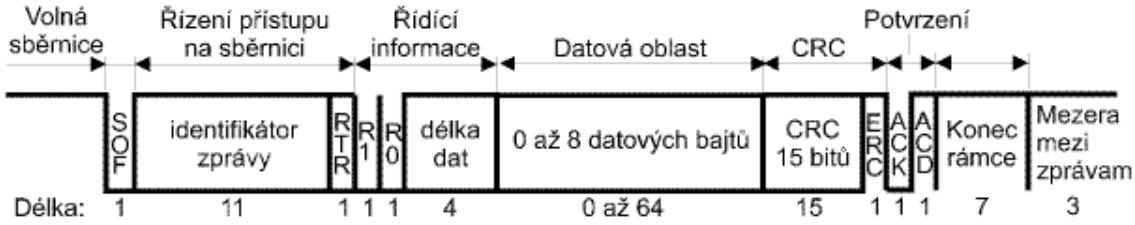
Datová zpráva (*Data Frame*)

Rozlišují se dva typy datových zpráv podle délky identifikátorů zpráv:

- Standardní formát zprávy (*Standard Frame*), specifikace CAN 2.0A – 11.bit identifikátor
- Rozšířený formát zprávy (*Extended Frame*), specifikace CAN 2.0B – 29.bit identifikátor

Oba typy zpráv mohou být vysílány na stejné sběrnici, pokud jsou použity v uzlech řadiče podle specifikace 2.0B.

Vyslání datové zprávy je možné pouze tehdy, je-li sběrnice volná (Bus Free). V případě, že uzel detekuje volnou sběrnice, začíná vysílat, úspěšnost vysílané zprávy záleží již na zmíněném řízení přístupu k médiu. Strukturu datové zprávy podle specifikace CAN 2.0A znázorňuje obr.4.



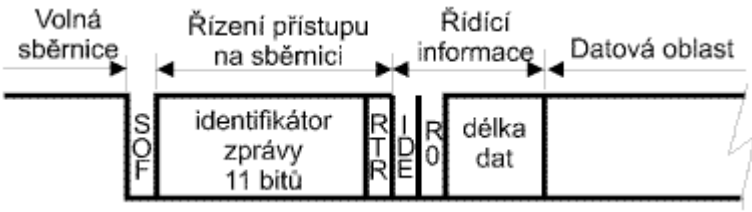
Obr.4 Datová zpráva podle specifikace CAN 2.0A viz.[12]

Význam jednotlivých částí datové zprávy podle specifikace 2.0A:

- **SOF** (Start of Frame) – 1 bit dominantní, začátek zprávy
- **Řízení přístupu na sběrnici** (Arbitration Field) – 12 bitů, určení priority zpráv
  - **Identifikátor zprávy –** 11bitů, udává význam přenášené zprávy
  - **RTR** (Remote request) – 1 bit, datová zpráva (dominantní) či žádost o data (recesivní)
- **Řídící informace** (Control Field) - 6 bitů
  - **R0, R1 –** 2 rezervované bity
  - **Délka dat** – 4 bity, počet přenášených datových bytů ve zprávě (0 až 8)
- **Datová oblast** (Data Field) – maximálně 8 bytů (64 bitů)
- **CRC** (CRC Field) – 16 bitů
  - **CRC** (Cyclic Redundancy Code) **–** 15 bitů, zabezpečovací CRC kód
  - **ERC** – 1 bit recesivní, CRC oddělovač
- **Potvrzení** (ACK Field) – 2 bity
  - **ACK** (Acknowledge) **–** 1 bit, potvrzení
  - **ACD** (Acknowledge Delimiter) – 1 bit recesivní, oddělovač potvrzení
- **Konec rámce** (End of Frame) – 7 bitů recesivních
- **Mezera mezi zprávami** (Interframe space) – 3 bity recesivní, oddělovač zpráv

Ve specifikaci CAN 2.0B je definován standardní a rozšířený formát datové zprávy. Doplňují se, nebo se mění následující bity datové zprávy:

- Standardní datový rámec
  - **IDE** (Identifier Extended) místo **R1 –** dominantní, standardní či rozšířený rámec



Obr.5 Začátek datové zprávy podle specifikace CAN 2.0B – standardní viz.[12]

- Rozšířený datový rámec
  - **IDE** (Identifier Extended) místo **R1 –** recesivní
  - **Řízení přístupu na sběrnici** – 31 bitů
    - je rozšířen o druhou 18-ti bitovou část identifikátoru
    - **SRR** (Substitute Remote Request) – recesivní, nahrazuje na konci 11-ti bitového identifikátoru **RTR**, zajišťuje kolizi mezi standardním a rozšířeným formátem, standardní formát má přednost
    - **RTR** (Remote Request) – umístěn na konci druhé části identifikátoru, udává zda se jedná o datovou zprávu či žádost o data. Priorita zpráv je určena na základě následujících bitů v pořadí: ID (11 bit), SRR, IDE, ID (18 bit), RTR.

Obr.6 Začátek datové zprávy podle specifikace CAN 2.0B – rozšířený viz.[12]

Žádost o data (Remote Frame)

Formát žádosti o data je velice podobný formátu datové zprávy. Pouze je zde RTR bit nastaven do recesivní úrovně a chybí zde datová oblast. Používá se jak standardní, tak i rozšířený formát zprávy (11-ti nebo 29-ti bitový identifikátor). Žádá-li nějaký uzel o zaslání dat, pak nastaví takový identifikátor zprávy, jako má datová zpráva, jejíž zaslání požaduje Tím je zajištěno, že pokud ve stejném okamžiku jeden uzel žádá o zaslání dat a jiný data se stejným identifikátorem vysílá, přednost v přístupu na sběrnici získá uzel vysílající datovou zprávu, neboť RTR bit datové zprávy je dominantní a tudíž má tato zpráva vyšší prioritu.

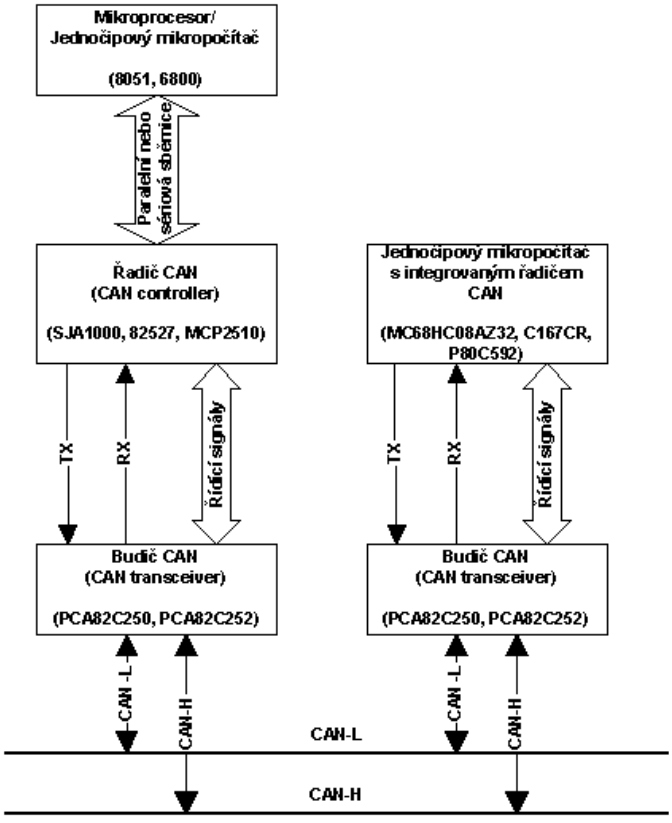
Chybová zpráva (Error Frame)

Chybová zpráva slouží k signalizaci chyb, které se vyskytnou na sběrnici CAN. V případě, že je uzlem detekována v přenášené zprávě chyba (chyba bitu, chyba CRC, chyba vkládání bitu, chyba rámce), je tímto uzlem vygenerován chybový rámec. Podle toho, v jakém stavu pro hlášení chyb se daný uzel nachází, generuje se buď aktivní (6 dominantních bitů) nebo pasivní (6 recesivních bitů) příznak chyby. Při generování aktivního příznaku chyby je přenášená zpráva poškozena, čímž i ostatní uzly začnou vysílat chybové zprávy.

Zpráva o přetížení (Overload Frame)

Zpráva o přetížení slouží k oddálení vysílání další zprávy. Tuto zprávu vysílá uzel, který ještě nestihl přijmout a zpracovat předchozí zprávu, tím je vysílání dalších zpráv pozdrženo. Formát zprávy o přetížení je stejný s formátem zprávy chybové.

1.3 Základní obvodové řešení sběrnice CAN



Obr. 7 Příklad současného typického zapojení viz.[11]

Popsané činnosti spojové vrstvy sběrnice CAN bývají realizovány výhradně hardwarově. Existuje zde mnoho různých řadičů sběrnice od řady výrobců. Řadič CAN (*CAN controller*) realizuje datovou spojovou vrstvu protokolu. Řadiče jsou buď realizovány jako samostatné periferie určené pro připojení k nějakému mikroprocesorovému systému, nebo jsou již přímo integrované uvnitř jednočipových mikropočítačů. Vedle řadiče je také zapotřebí budič CAN (*CAN transceiver*), který realizuje fyzickou vrstvu protokolu CAN, tj. převod signálu z řadiče (úroveň TTL) na logické úrovně sběrnice CAN. Na obrázku obr.7 je zobrazen příklad současného typického zapojení.

Elektronické součástky pro CAN

Přehled vybraných typů elektronických součástek – budičů, řadičů a jednočipových mikropočítačů pro sběrnici CAN.

- **Budič sběrnice** (CAN transceiver)
  Nejrozšířenější jsou budiče firmy Philips, avšak existují i jiní výrobci.
  - CAN high-speed – PCA82C250, jeho novější náhrada TJA1050, PCA82C251 pro nákladní vozy a autobusy
  - CAN low-speed – PCA82C252, jeho novější náhrada TJA1053 / TJA1054, případně budič Infineon TLE6252G
- **Řadič sběrnice** (CAN controller)
  Není-li řídící jednotka vybavena jednočipovým mikropočítačem se zabudovaným CAN řadičem, musí používat externí řadič zapojený k mikroprocesoru. Zapojení je většinou provedeno pomocí datové a adresové sběrnice, avšak některé řadiče jsou vybaveny také sériovým rozhraním (např. rozhraní SPI). Následuje přehled několika rozšířených řadičů:

  - Intel 82C257 – má i sériové rozhraní
  - Philips SJA1000 (náhrada za 82C200)
  - Microchip MCP2510 – pouze sériové rozhraní
  - Infineon 81C90/91 – má i sériové rozhraní
  - OKI MSM9225 – má i sériové rozhraní

- **Jednočipové mikropočítače s integrovaným CAN řadičem**

Počet typů jednočipových mikropočítačů s integrovaným řadičem CAN neustále roste, zde je uvedeno několik typů:

- Infineon C164CI, C167CR
- Microchip PIC18C658/858, PIC8F248/258/448/458
- Motorola 68HC05X32/X16, 68HC08Azx, 68HC908AZ60
- Philips 8x592, 8x591, XA-C3
- Dallas DS80C390 – 2 vestavěné řadiče CAN
- Fujitsu MB90F598, MB90F594 (2 CAN řadiče), MB91F361 (3 CAN řadiče)
- Hitachi H8/300H
- Atmel AT90CAN128
- Intel 87C196CA/CB

2 Hardwarová realizace uzlu CAN

2.1 Popis použitých komponent uzlu CAN

Výběr konkrétních komponent pro vlastní hardwarovou realizaci uzlu byl volen s ohledem na jednoduchost a funkčnost tohoto zařízení. Neméně důležitým faktorem byla také jejich cena a dostupnost. V této kapitole je uveden stručný popis základních vlastností jednotlivých komponent.

2.1.1 Mikrokontrolér Atmel AT90S8535

Jedná se nízkopříkonový osmibitový mikrokontrolér založený na AVR RISC architektuře (s redukovaným instrukčním souborem), tzn.že provádí jednotlivé instrukce v jediném hodinovém cyklu, čím se stávají jednodušší a mnohem výkonnější. Instrukční soubor obsahuje 118 instrukcí. K dispozici je 32 osmibitových registrů. Paměť programu je tvořena zabudovanou FLASH pamětí s kapacitou 8kB, kterou je možno programovat klasickým paralelním programátorem, nebo přímo v systému pomocí rozhraní SPI (*Serial Peripheral Interface*), počet garantovaných přeprogramování je 1000 cyklů. Datová paměť je buď typu RAM (512B) nebo EEPROM (rovněž 512B). Počet vstupů/výstupů je 32, rozdělené jsou do 4 portů po osmi vývodech. Každý z nich je možno nastavit jako vstupní, či výstupní. Obvod dále disponuje dvěma 8-bitovými čítači/časovači a jedním 16-bitovým. Další důležitou vlastností je existence zabudovaného asynchronního sériového kanálu (UART). Obvod obsahuje i řadu dalších užitečných vlastností, které ovšem nejsou pro tuto realizaci využity, jako např. 8-kanálový 10-bitový A/D převodník, či WDT (watch dog timer). Mikrokontrolér je dodáván v pouzdře DIP40, alternativně také v PLCC44, TQFP44, MLF44.

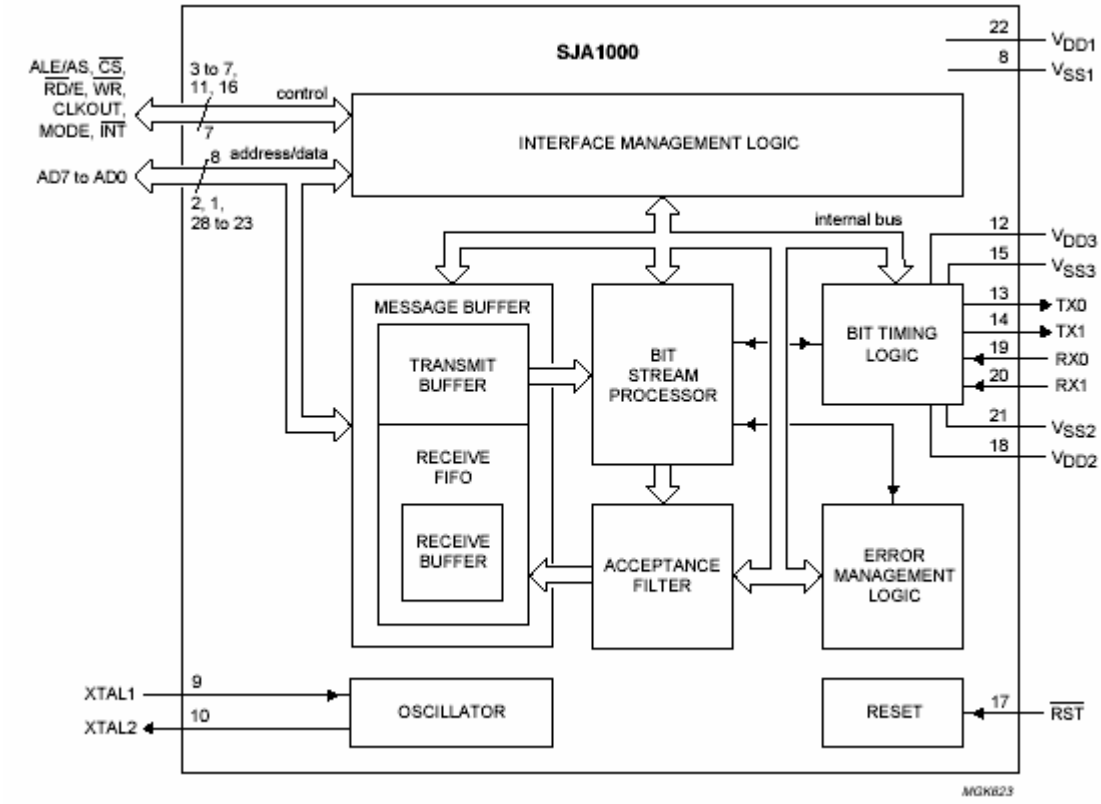
2.1.2 Řadič CAN sběrnice Philips SJA1000

Řadič CAN sběrnice SJA1000 od firmy Philips, je rozšířenější náhradou za variantu PCA82C200. Tato varianta umožňovala komunikaci pouze podle specifikace CAN 2.0A, v rozšířeném formátu zprávy (specifikace CAN 2.0B) pracovala v tzv. pasivní módu (ignoruje, ale toleruje výskyt 29-bitového identifikátoru). Z důvodu zachování vývodové a elektrické kompatibility s tímto předchůdcem je obvod SJA1000

vybaven dvěma módy: BasicCAN a PeliCAN. Funkce řadiče v BasicCAN módu jsou totožné s obvodem PCA82C200, v PeliCAN módu pracuje řadič podle specifikace CAN 2.0B (se standardním i rozšířeným formátem zprávy). V tomto módu existuje jiná struktura registrů a nabízejí se možnosti využití dalších funkcí, jako jsou např. detekce různých chyb, či rozšířené filtrování příchozích zpráv. Maximální možná přenosová rychlost je 1Mbit/s. Operační rozsah teplot je v rozmezí –40°C až +125°C. Obvod se dodává v pouzdrech DIP28 nebo SO28 pro SMT.

Blokový diagram SJA1000

Přístup ke kontroléru SJA1000 je paralelní s oddělenou adresovou/datovou a řídící sběrnicí. Obvod je doplněn o externí krystal s frekvencí 24MHz. Propojení s řídícím mikropočítačem je uvedeno v kapitole 2.2.1.



Obr. 8 Blokový diagram obvodu SJA1000 viz.[4]

Stručný popis jednotlivých částí blokového diagramu obvodu SJA100

- Interface Management Logic – zajišťuje komunikaci mezi bloky, řídí vnitřní sběrnici
- Message Buffer – paměť pro uložení přijímaných (*Receive Buffer*) a vysílaných (*Transmit Buffer*) zpráv

- Bit Stream Processor – třídí data mezi Message Bufferem a CAN sběrnicí
- Bit Timing Logic – zajišťuje připojení CAN rozhraní (přes budič sběrnice PCA82C250), výstup je v TTL logice
- Acceptance Filter – filtruje příchozí zprávy
- Error Management Logic – umožňuje detekci chybových stavů
- Oscilator – připojení krystalu (24MHz)
- Reset – reset obvodu (aktivní log.0)

2.1.3 Budič CAN sběrnice Philips PCA82C250

Budič CAN sběrnice PCA82C250 od firmy Philips realizuje převod signálu z řadiče CAN SJA1000 (TTL úrovně) do fyzické vrstvy CAN. Je plně kompatibilní se standardem ISO 11898. Přenosová rychlost je možná až do frekvence 1MHz. Je redukována možnost rádiové interference a interference s elektromagnetickým rušením. Obvod je vybaven tepelnou ochranou a má nízkou spotřebu v pasivním módu. Realizuje připojení až 110 uzlů CAN současně. Obvod je vyráběn v pouzdrech DIP8 nebo SO8.

2.1.4 Sériové rozhraní Maxim MAX232CPE

Obvod MAX232 realizuje převod signálů z mikropočítače v úrovni TTL (0 a 5V) na úroveň sériového rozhraní RS232 (±10V). Obvod je doplněný o 4 kondenzátory pro nábojovou pumpu, která z napájecího napětí vytvoří potřebná napětí +10V a –10V. Napájecí napětí je 5V, maximální přenosová rychlost je 120kbit/s. Obvod je vybaven dvěma vysílači a dvěma přijímači. Je dodáván v pouzdrech DIP16 a SOIC16.

2.1.5 Obvod pro ovládání krokových motorků Philips SAA1027

Obvod SAA1027 je bipolární integrovaný obvod pro ovládání čtyřfázových krokových motorků. Je vysoce odolný proti vstupnímu rušení. Otáčení motorku je ovládáno přes vstup *Count Input*, kde s každou náběžnou hranou se motorek otočí o jeden krok. Rychlost otáčení je úměrná frekvenci vstupních impulsů. Je umožněna funkce přepínání směru otáčení motorků ve směru i protisměru hodinových ručiček (vstup *Mode*) a možnost resetování (vstup *Reset*). Maximální hodnota napájecího napětí i napětí na řídících vstupech je 18V, maximální výstupní proud je 500mA a rozsah operačních teplot je –20°C až +70°C.

2.1.6 Krokový motor

Krokové motory mají v současné době širokou škálu použití. Vzhledem k tomu, že jsou nejjednodušším akčním členem pro převod digitálního signálu na polohu, resp. úhel natočení, nacházejí celou řadu uplatnění v oblasti řídící, výpočetní a regulační techniky. Jejich nejčastější výskyt je u různých typů tiskáren, zapisovačů a elektrických posuvných zařízení. V oblasti řídící a regulační techniky se využívají spolu s převodovkou na změnu mechanických poloh ventilů, směšovačů, nebo pro posuvy u menších NC strojů atd.

Bylo řečeno [Rydlo, 2000], že krokový motor je impulsně napájený motor, jehož funkční pohyb je nespojitý a děje se po jednotlivých úsecích (krocích). Uvnitř krokového motoru je generováno pulsující magnetické pole postupným napájením jednotlivých pólových dvojic stejnosměrným proudem. Počet stabilních poloh motoru je dán počtem kroků motoru na jednu otáčku. K řízení krokového motoru slouží ovladač krokového motoru, který musí splnit požadavky na výkonové buzení jednotlivých fází a vytvořit požadovanou časovou posloupnost buzení fází motoru. Otáčky krokového motoru jsou úměrné kmitočtu řídícího signálu, určí se podle následujícího vztahu:

$$n = \frac{60.f_k.\alpha}{360}$$

kde: n je počet otáček za minutu

f_k je kmitočet kroků v Hz

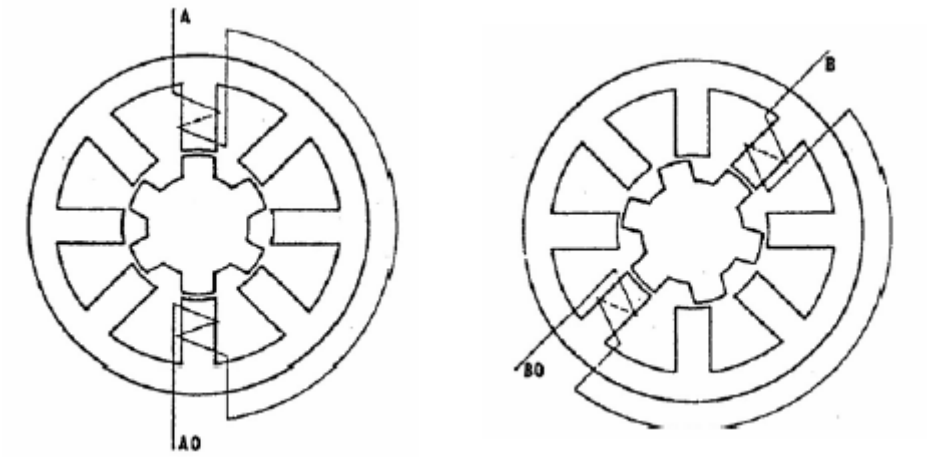
α je velikost kroku ve stupních

Krokové motory se podle konstrukčního uspořádání dělí do tří skupin:

- Krokové motorky s **pasivním rotorem** (reluktanční) – rozdílný počet pólů na statoru (póly s navinutými cívkami) a rotoru (bez vinutí).
- Krokové motorky s **aktivním rotorem** – počet pólů na statoru a rotoru (tvořen permanentním magnetem) je soudělný. Dělí se motory s radiálně a axiálně polarizovanými permanentními magnety
- Krokové motorky **hybridní** – slučují konstrukční principy obou typů

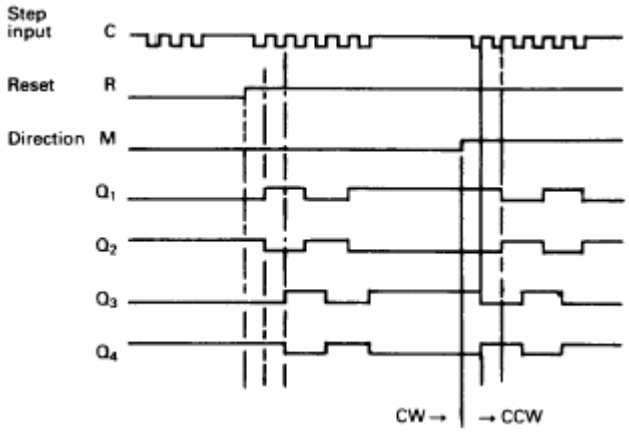
Krokové motory se dále dělí podle počtu fází. V tomto případě se jedná o ovladač SAA1027 sloužící pro ovládání čtyřfázových krokových motorků.

Princip činnosti čtyřfázového krokového motorku s pasivním rotorem je znázorněna na následujícím obrázku.



Obr. 9 Otáčení krokového motorku v závislosti na přepínání buzení jednotlivých fází vinutí viz.[7]

Otáčení motorku se provádí přepínáním buzení jednotlivých fází vinutí. Z obrázku obr.9 je patrné, že nejprve je buzeno vinutí A, poté vinutí B, čímž dojde k pootočení motorku o jeden krok. Řízení směru otáčení se provádí změnou pořadí buzení jednotlivých vinutí. Řízení posloupnosti kroků je patrné na následujícím obrázku odpovídajícímu již zmíněnému obvodu SAA1027.



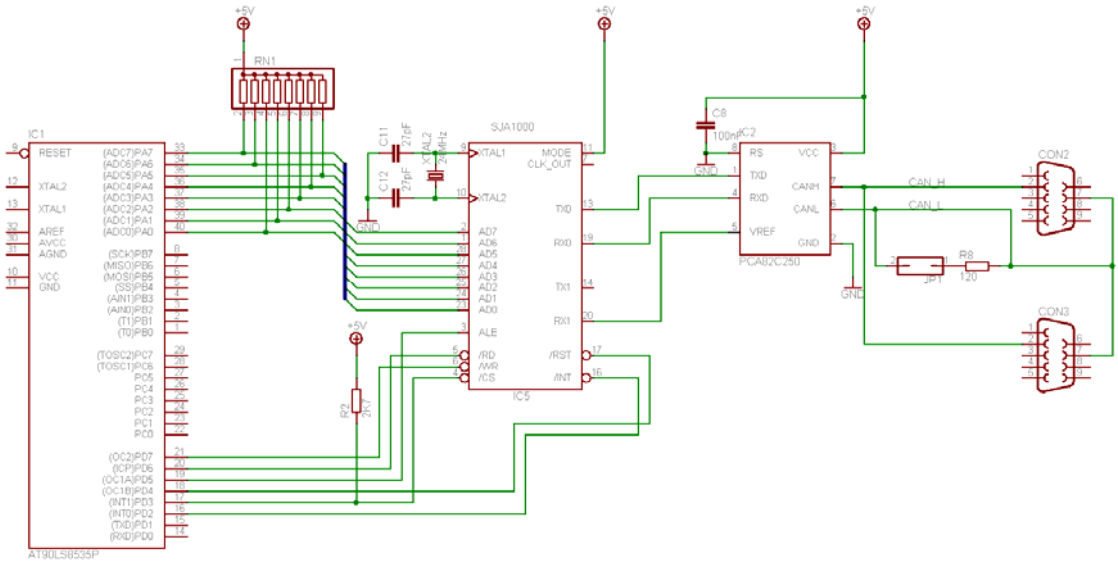
Obr.10 Časový diagram ovládání krokového motorku obvodem SAA1027 viz.[8]

2.2 Stručný popis schéma zapojení

Celkové schéma zapojení včetně návrhu desky plošných spojů je uvedeno příloze, v této kapitole bude popsáno pouze vzájemné propojení nejdůležitějších obvodů.

2.2.1 Realizace připojení na CAN sběrnici

Jak již bylo uvedeno dříve, tak část obvodu pro komunikaci po sběrnici CAN je tvořena řadičem (SJA1000) a budičem (PCA82C250) sběrnice CAN. Řadič je připojen přímo k mikrokontroléru AT90S8535, jeho propojení je zřejmé z následujícího obrázku.



Obr. 11 Zapojení obvodu pro komunikaci po CAN sběrnici

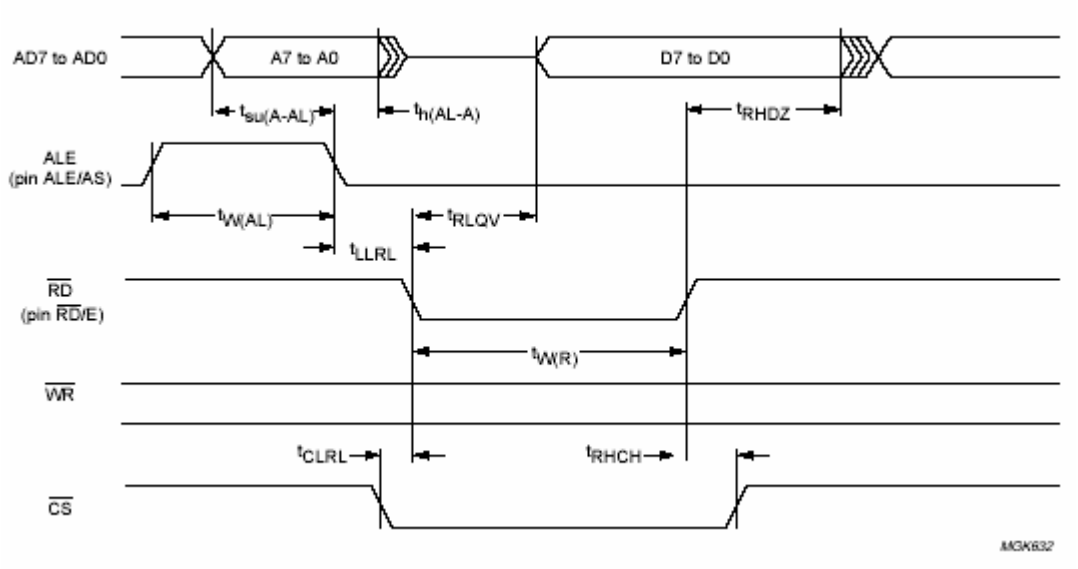
Řadič SJA1000 je vybaven externím krystalem 24MHz připojeným k vývodům XTAL1 a XTAL2. Vývodem MODE se volí komunikace v Intel módu (log.1) nebo v Motorola módu (log.0), z toho důvodu je připojen na 5V. Adresová/datová sběrnice řadiče je přímo propojena s portem PA procesoru s použitím zdvihových rezistorů RN1. Řídící sběrnice je na portu PD a tvoří ji vývody ALE, $\overline{CS}$, $\overline{\mathrm{RD}}$, $\overline{\mathrm{WR}}$. Reset ($\overline{RST}$) řadiče je ovládán programově pomocí vývodu PD4. Z řadiče je dále vyveden výstup přerušení $\overline{INT}$ přivedený na vstup externího přerušení procesoru PD2 ($INT0$). Na vývodu CLKOUT je možno nastavit frekvenci rovnou:

$$f_{OSC}, \frac{f_{OSC}}{2}, \frac{f_{OSC}}{4}, \frac{f_{OSC}}{6}, \frac{f_{OSC}}{8}, \frac{f_{OSC}}{10}, \frac{f_{OSC}}{12}, \frac{f_{OSC}}{14}.$$

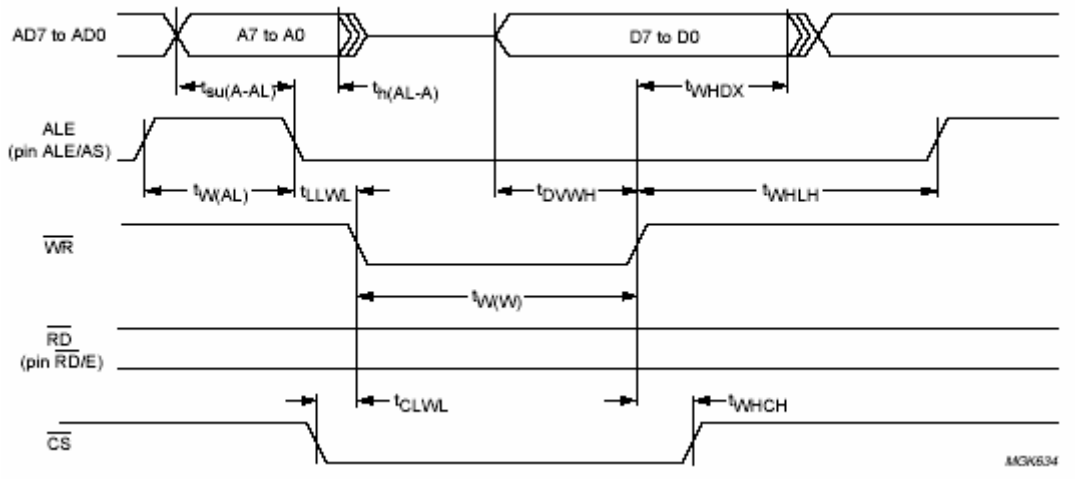
Význam jednotlivých řídících signálů

- *ALE* – signál pro zápis adresy/dat
- $\overline{CS}$ – (*chip select*) povoluje přístup k řadiči
- $\overline{RD}$ – (*read enable*) povoluje čtení dat z řadiče
- $\overline{WR}$ – (*write enable*) povoluje zápis dat do řadiče

Následující časové diagramy znázorňují průběhy řídících signálů při čtení, resp. zápisu dat z/do řadiče SJA1000.



Obr. 12 Časový průběh čtení dat z SJA1000 viz.[4]

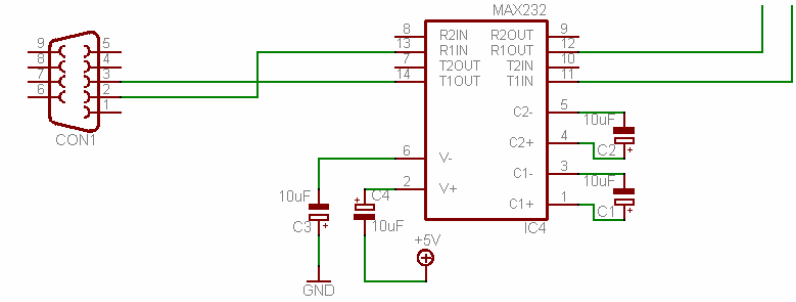


Obr. 13 Časový průběh zápisu dat do SJA1000 viz.[4]

Způsob připojení budiče CAN sběrnice PCA82C250 je rovněž patrný z obr.11. Budič je propojen pomocí dvou vodičů TXD a RXD na vývody TX0 resp. RX0 řadiče. Vývod řadiče RX1 je propojen s referenčním napětím VREF 2,5V od budiče. Vývody CANH a CANL jsou již vyvedeny na konektory D-SUB, kterými je již možné přímé připojení na CAN sběrnici. Pomocí jumperu JP1 je možno připojit zakončovací odpor sběrnice R8 s danou hodnotou 120Ω.

2.2.2 Připojení k PC po RS232

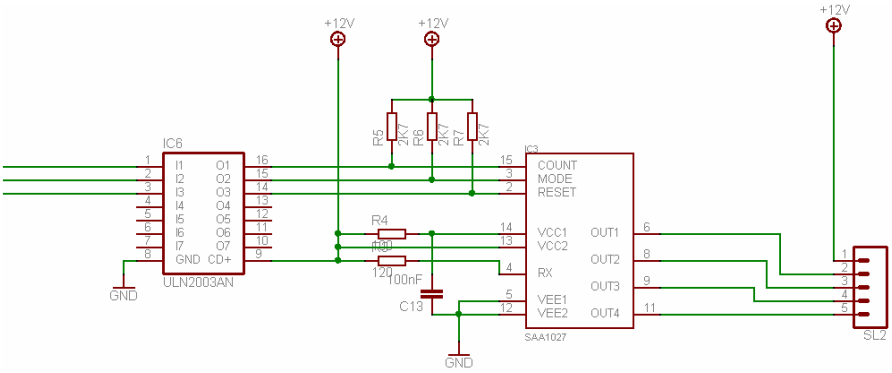
Připojení CAN uzlu k PC je realizováno pomocí obvodu MAX232CPE. Výstup z tohoto obvodu je přes konektor D-SUB propojen se sériovým konektorem na PC. Zapojení tohoto obvodu je znázorněno na následujícím obrázku.



Obr. 14 Zapojení obvodu MAX232CPE

2.2.3 Realizace obvodu pro ovládání krokových motorků

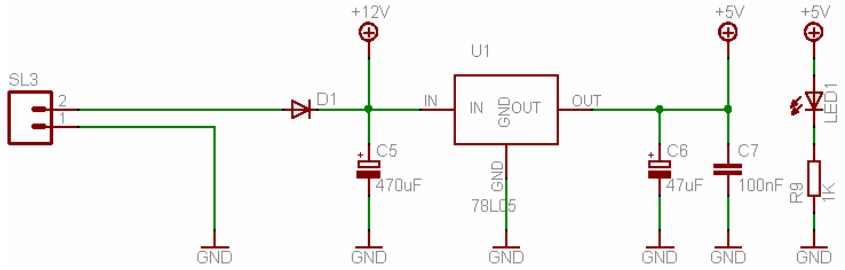
Ovládání krokových motorků je provedeno obvodem SAA1027 (viz. 2.1.4). Řídící signály (*Reset*, *Mode* a *Count*) jsou vyvedeny z portu PC procesoru (PC0, PC1 a PC2). Tyto je nutné přes obvod ULN2003A (DIP16), který obsahuje sedm Darlingtonových zapojení tranzistorů, přivést na příslušné vstupy obvodu SAA1027. Použity jsou zdvihové odpory 2k7. Odpor R5 120Ω zapojený na vstup RX slouží k nastavení hodnoty výstupního proudu. Zapojení obvodu je znázorněno na následujícím obrázku.



Obr. 15 Zapojení obvodu pro ovládání krokových motorků

2.2.4 Napájení obvodu

Napájení obvodu je z důvodu ovládání krokového motorku nutné 12V, avšak ostatní obvody jsou napájeny 5V. Proto je na desce vytvořen regulátor napětí na 5V, včetně signalizace pomocí LED diody. K tomuto je využit obvod 7805 (LM340), jehož zapojení je znázorněno na následujícím obrázku.



Obr. 16 Realizace napájení obvodu

2.2.5 Programátor SPI

Programování obvodu je umožněno přímo v zapojení (ISP - *In System programming*) pomocí programátoru SPI, přes sériovou linku z PC, zhotoveného přímo v konektoru. Propojení s procesorem je přes pěti-vodičový konektor na vývody PB5 (MOSI), PB6 (MISO) a PB7 (SCK). Dalšími vývody jsou Reset a GND. K procesoru je dále připojen krystal 8MHz.

3 Popis jednotlivých programů

V této kapitole jsou uvedeny základní funkce a principy jednotlivý programů, jak pro řídící mikrokontrolér, tak i pro nastavování konfigurace z PC a program pro ověření funkce. Nakonec je uvedena možnost programování v aplikační vrstvě.

3.1 Řídící program mikrokontroléru uzlu CAN

Řídící program je napsán v programovacím jazyce C ve vývojovém prostředí CodeVisonAVR C, v kterém se následně provede i kompilace do formátu *.hex. Vlastní naprogramování mikroprocesoru se provádí pomocí programu PonyProg2000, přes rozhraní SPI přímo v zapojení (ISP – *In System Programming*). Programátor, vytvořený z diskrétních součástek, je integrovaný do konektoru sériového kabelu.

Hlavní funkcí programu je přijímat zprávy přicházející po CAN sběrnici a na jejich základě provádět požadované akce krokového motorku. Zpět se posílá stav motorku po ujetí požadovaných kroků. Mikrokontrolér dále zjišťuje, zda se nevyskytla žádost z PC o změnu konfigurace uzlu. V tom případě se uloží přijaté hodnoty po RS232 do EEPROM uvnitř mikrokontroléru a provede se inicializace uzlu. Následuje detailnější popis řídícího programu, nastavení počátečních parametrů, vysvětlení jednotlivých funkcí a činnosti hlavního programu.

3.1.1 Nastavení použitých registrů

Na začátku programu je nejprve nutno nastavit směr a vlastnosti vývodů jednotlivých portů. Mikroprocesor je vybaven čtyřmi paralelními porty (PA, PB, PC, PD) po osmi vývodech, majících stejné vlastnosti. Chování vývodů portu PB jsou určeny nastavením registrů DDRB (určuje směr toku dat) a PORTB (datový registr). Viz. Tab. 1.

DDBn	PORTBn	vstup/výstup	pull-up	stav vývodu
0	0	vstup	ne	třístavový vstup (pull-up odpojen)
0	1	vstup	ano	pull-up připojen
1	0	výstup	ne	log.0
1	1	výstup	ne	log.1

Tab. 1 Vliv DDBn a PORTBn na chování vývodu PBn

Na portu PB jsou umístěny tři tlačítka (PB.0, PB.1 a PB.2) pro přímé ovládání krokového motorku. Je tedy nutno je nastavit jako vstupní se zvyšovacím odporem (*pull-up*). Proto příslušné bity registru DDRB jsou nataveny na log.0 a portu PORTB na log.1. Obdobně se určí vlastnosti i ostatních portů. Z portu PC jsou vyvedeny řídící signály pro ovládání krokového motorku (reset,směr a hodinové impulsy). Tento port je tedy zapojen jako výstupní, což určuje registr DDRC (0xFF). Na portu PD jsou zapojeny řídící signály pro řadič CAN, přerušení od řadiče a dva vývody (RXD a TXD) pro sériovou linku RS232. Všechny řídící signály jsou nastaveny jako výstupní, přerušení od řadiče CAN je nastaveno jako vstupní se zvyšovacím odporem. Směr vývodů pro sériový kanál není třeba nastavovat, neboť zde probíhá obousměrná komunikace a příslušné nastavení si zajistí procesor sám. Registr DDRD má tedy hodnotu 0xF8 a registr PORTD 0x04. Zbývá ještě port PA, na kterém je vyvedena adresová a datová sběrnice pro řadič CAN. Jedná se o obousměrnou komunikaci, proto směr toku dat není možno předem definovat, jeho nastavení se provede až při každém volání příslušné funkce pro čtení, resp. zápis.

Dále je nutné nastavit parametry komunikace po sériovém kanálu (UART). K tomuto účelu jsou zde registry UCR (řídící registr) a UBRR (registr přenosové rychlosti). Řídící registr UCR (viz.Obr.15) povoluje příjem a vysílání a obsahuje masky přerušení.

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8

Obr. 17 Registr UCR

Význam jednotlivých bitů registru UCR

- RXCIE – povolení přerušení po dokončení příjmu (log.1)
- TXCIE – povolení přerušení po dokončení vysílání (log.1)
- UDRIE – povolení přerušení při vyprázdnění datového registru (log.1)
- RXEN – příjem povolen (log.1)
- TXEN – vysílání povoleno (log.1)
- CHR9 – datový znak je dlouhý 9 bitů (log.1)
- RXB8 – devátý přijatý bit (je-li CHR9 = 1)
- RXB8 – devátý vysílaný bit (je-li CHR9 = 1)

Registr UCR je v této aplikaci nastaven na hodnotu 0x98, což odpovídá nastavení bitů RXCIE, RXEN a TXEN do log.1. To znamená, že je povolen příjem a vysílání znaků a je povoleno přerušení po dokončení příjmu znaku.

Druhým registrem je registr přenosové rychlosti UBRR. Nastavením tohoto registru na hodnotu 0x33 (51) získáme výslednou přenosovou rychlost cca. 9600 Bd (s chybou 0,2%) dle vzorce:

$$PR = \frac{f_0}{16 \cdot (UBRR + 1)} = \frac{8.10^3}{16 \cdot (51 + 1)} = 9615{,}4\ Bd$$

kde f_0 je kmitočet krystalu mikrokontroléru, tedy 8MHz.

Je nutné ještě nastavit časovač pro generování hodinových impulsů pro řízení rychlosti otáčení krokového motorku. Mikrokontrolér AT90S8535 disponuje dvěmi 8bitovými čítači/časovači a jedním 16bitovým. Z důvodu většího možného rozsahu generovaných frekvencí je zvolen 16bitový čítač/časovač 1. Řídícími registry čítače/časovače 1 jsou TCCR1A a TCCR1B. Registr TCCR1A slouží pro konfiguraci režimů Output Compare a PWM generátoru. Jelikož tyto režimy nejsou v programu využity, je hodnota v registru TCCR1A rovna 0x00. Uvedena je pouze struktura tohoto registru bez nějakého bližšího vysvětlení.

7	6	5	4	3	2	1	0
COM1A1	COM1A0	COM1B1	COM1B1	x	x	PWM11	PWM10

Obr. 18 Registr TCCR1A

Registr TCCR1B slouží pro výběr hodinového zdroje čítače/časovače 1 a konfiguraci režimů Input Capture a Output Compare.

7	6	5	4	3	2	1	0
ICNC1	ICES1	x	x	CTC1	CS12	CS11	CS10

Obr. 19 Registr TCCR1A

Bity ICNC1, ICES1 a CTC1 se opět týkají režimů Input Capture a Output Compare, tudíž jsou nastaveny na log.0. Zbývající bity CS12 CS11 a CS10 slouží k výběru hodinového signálu. Jejich význam je naznačen v následující tabulce.

CS12	CS11	CS10	popis
0	0	0	stop, čítač/časovač1 je odstaven
0	0	1	f_0 (hodinový signál mikrkontroléru CLK)
0	1	0	$f_0/8$ (1/8 CLK)
0	1	1	$f_0/64$ (1/64 CLK)
1	0	0	$f_0/256$ (1/256 CLK)
1	0	1	$f_0/1024$ (1/1024 CLK)
1	1	0	sestupná hrana T1
1	1	1	náběžná hrana T1

Tab. 2 Výběr hodinového signálu čítače/časovače 1

V tomto případě je předdělička hodinového signálu rovná 1/256, tedy CS12 = 1, CS11 = 0 a CS10 = 0, potom hodnota v registru TCCR1B je 0x04.

Čítač/časovač 1 je představován párem 8bitových registrů TCNT1H a TCNT1L pracujících jako volně běžící 16bitový čítač čítající nahoru. Pro získání požadové hodnoty časové prodlevy, je nutné tyto registry přednastavit na určitou hodnotu, od které se budou dopočítávat do 0xFFFF (65535). Nastavení těchto registrů se pro určitou požadovanou periodu vypočte dle vzorce (příklad pro získání periody 1s):

$$TCNT1H, TCNT1L = 65535 - \frac{f_0}{pred \cdot f} = 65535 - \frac{8.10^6}{256 \cdot 1/1} = 34285 \quad (0x85, 0xED)$$

kde *pred* je hodnota předděličky nastavena v registru TCCR1B. Tyto hodnoty jsou uloženy v proměnných *RegH* a *RegL*. Po načítání hodnot registrů TCNT1H a TCNT1L do hodnoty 0xFFFF (65535) se vyvolá přerušení od čítače/časovače 1, ovšem pokud je povoleno, a zároveň je povoleno i globální přerušení. Globální přerušení se povolí na začátku hlavního programu hned po inicializační části nastavením bitu *I* stavového registru na log.1 (I = 1).

Povolení přerušení od čítače/časovače 1 se nastaví na příslušných místech programu (viz. dále) pomocí registru TIMSK představujícího masku přerušení.

7	6	5	4	3	2	1	0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	x	TOIE0

Obr. 20 Registr TIMSK

Čítače/časovače 1 se týká bit TOIE1, jehož nastavením do log.1 se povolí přerušení při jeho přetečení. Bude-li tedy hodnota v tomto registru rovna 0x04, bude příslušné přerušení povoleno, při hodnotě 0x00 bude zakázáno. Význam ostatních bitů není třeba blíže vysvětlovat, neboť na jejich nastavení není v programu brán zřetel.

3.1.2 Popis vytvořených funkcí

V této části jsou popsány jednotlivé vytvořené funkce, které jsou volány z hlavního programu. Nejprve jsou zmíněny funkce týkající se nastavení parametrů řadiče CAN sběrnice, dále jsou posány funkce příjmu a vysílání datových zpráv po CAN a nakonec je vysvětlena činnost přerušovacích rutin.

Pro přístup k registrovému prostoru CAN řadiče SJA1000 jsou vytvořeny funkce pro čtení a zápis, nacházející se v knihovně *CANinout.h*. Tyto funkce jsou vytvořeny podle časových průběhů řídících signálů pro čtení a zápis, ktreré jsou znázorněny v kapitole 2.2.1 na Obr.10 resp. Obr.11.

- **void** outport **(unsigned char** adr**, unsigned char** data**)** – zápis dat (data) na adresu (adr) řadiče CAN SJA1000

- **unsigned char** inport **(unsigned char** adr**)** – vrací přečtená data z adresy (adr) řadiče CAN SJA1000

Inicializace CAN sběrnice

Pro nastavení parametrů sběrnice CAN je vytvořena funkce *InitCAN()*, ve které jsou odkazy na funkce *SetParameter()* a *InitInterrupt()*.

- **void** InitCAN **(void)** – inicializace CAN sběrnice

Nastavení parametrů se provádí v tzv. resetovacím módu, proto se nejprve musí nastavit registr *mode* (adresa 0) na 0x01 a po vlastním nastavení parametrů uvést zpět do operačního modu hodnotou 0x00.

- **void** SetParameter **(void)** – nastavení registrů řadiče CAN

V této funkci dochází k nastavení parametrů řadiče CAN, jako je přenosová rychlost komunikace (baudrate), výstupní mód (normální), režim činnosti řadiče a

filtrování přijímaných zpráv. Přenosová rychlost je uložena v EEPROM v proměnné *eebaud*, jako číselná hodnota. Podle hodnoty v této proměnné dojde k vlastnímu nastavení příslušných registrů časování bitů BTR0 a BTR1 (*Bus Timing Registr 0* a *Bus Timing Registr 1*).

7	6	5	4	3	2	1	0
SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

Obr. 21 Bus Timing Registr 0 (BTR0) – adresa 6

Registr BTR0 definuje systémové hodiny CAN pomocí bitů BRP.0 až BRP.5 (*Baud Rate Prescaler*) dle následujícího vzorce:

$$t_{scl} = 2 \cdot t_{CLK} \cdot (32 \cdot BRP.5 + 16 \cdot BRP.4 + 8 \cdot BRP.3 + 4 \cdot BRP.2 + 2 \cdot BRP.1 + BRP.0 + 1)$$

kde t_{CLK} je perioda frekvence vnějšího krystalu XTAL $t_{CLK} = \frac{1}{f_{XTAL}}$.

Dále je tímto registrem také definována šířka synchronizačního skoku SJW (*Synchronization Jump Width*) dle vzorce:

$$t_{SJW} = t_{scl} \cdot (2 \cdot SJW.1 + SJW.0 + 1)$$

Registr BTR1 definuje dobu periody jednoho bitu, umístění vzorkovacího bodu a počet těchto vzorkovacích bodů v jedné periodě.

7	6	5	4	3	2	1	0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

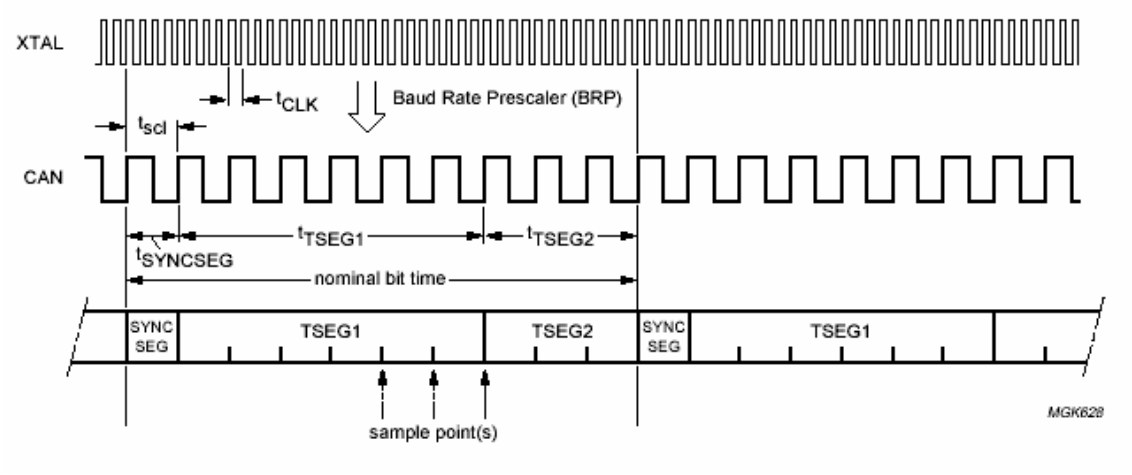
Obr. 22 Bus Timing Registr 1 (BTR1) – adresa 7

Hodnota bitu SAM udává počet vzorkovacích bodů (*sample point*), log.1 odpovídá trojnásobnému vzorkování (doporučeno pro nižší přenosové rychlosti), log.0 odpovídá jednomu vzorkovacímu bodu. Časový segment 1 TSEG1 (*Time Segment 1*) a časový segment 2 TSEG2 (*Time Segment 2*) určuje počet hodinových cyklů v jedné periodě bitu a umístění vzorkovacího bodu, kde platí:

$$t_{SYNCSEG} = 1 \cdot t_{scl}$$

$$t_{TSEG1} = t_{scl} \cdot (8 \cdot TSEG1.3 + 4 \cdot TSEG1.2 + 2 \cdot TSEG1.1 + TSEG1.0 + 1)$$

$$t_{TSEG2} = t_{scl} \cdot (4 \cdot TSEG2.2 + 2 \cdot TSEG2.1 + TSEG2.0 + 1)$$



Obr. 23 Časování bitu viz.[4]

V programu jsou pro konkrétní hodnoty přenosových rychlostí (10kBit, 20kBit, 50kBit, 100kBit, 125kBit, 250kBit, 500kBit, 800kBit a 1Mbit) nastaveny příslušné hodnoty do registrů BTR0 a BTR1. Vzorkovací bod je pro nižší přenosové rychlosti (do 500kBit) nastaven jako trojnásobný, pro vyšší (od 800kBit) pouze jednoduchý.

V registru nastavení výstupu OCR (*Output Control Register*) je zvolena kombinace bitů odpovídající normálnímu výstupnímu módu. Jedná se o hodnotu 0x1A. Na následujícím obrázku je uvedena struktura tohoto registru.

7	6	5	4	3	2	1	0
OCTP1	OCTN1	OCPOL1	OCTP0	OCTN0	OCPOL0	OCMODE1	OCMODE0

Obr. 24 Output Control Registr (OCR)- adresa 8

Dalším registrem, který je nutno nastavit je registr výstupního hodinového signálu CDR (*Clock Divider Register*). V tomto registru je vedle nastavení frekvence možného výstupního signálu na vývodu CLKOUT, také možnost vybrat mezi základním (*BasicCAN*) módem a rozšířeným (*PeliCAN*) módem. Jak již bylo uvedeno v kapitole 2.1.2, tak BasicCAN mód pracuje pouze s 11ti bitovým identifikátorem dle specifikace CAN 2.0A, zatímco PeliCAN mód může pracovat jak s tímto, tak i s rozšířeným 29ti bitovým identifikátorem dle specifikace CAN 2.0B.

7	6	5	4	3	2	1	0
CAN mode	CBP	RXINTEN	0	clock off	CD.2	CD.1	CD.0

Obr. 25 Clock Divider Registr (CDR) - adresa 31

Zde je důležité právě nastavení bitu CAN mode na log.1, pro zvolení PeliCAN módu. Nastavení konkrétní hodnoty výstupní frekvence na vývodu je možno zjistit z tabulky (Tab.10) umístěné v příloze. Další dva bity již nejsou využity.

Následuje nastavení filtru přijímaných zpráv (*Acceptance Filter*). V řadiči CAN sběrnice SJA1000, existují dva režimy filtrování – jednoduchý (*single*) a dvojitý (*double*). V této aplikaci je použit jednoduchý režim nastavením bitu MOD.3 (*Acceptance Filter Mode*) v *mode* registru na log.1. Nastavují se čtyři registry AcceptCode (ACR0 – ACR3) a AcceptMask (AMR0 – AMR3). Každá zpráva přijímaná řadičem SJA1000 prochází nejprve tímto filtrem, ve kterém jsou porovnány bity identifikátoru zprávy s bity v registru AcceptCode. Obsah registru AcceptMask určuje které bity se budou porovnávat. Porovnávají se jen ty bity ve kterých je v registru AcceptMask log.0. V případě, že identifikátor zprávy neodpovídá zadanému filtru je zpráva ignorována.

	MSB							LSB	MSB							LSB
ACR	1	1	1	0	0	0	0	1	1	1	0	0	x	x	x	x
AMR	0	0	0	0	1	1	1	1	0	0	0	0	x	x	x	x
ID	1	1	1	0	x	x	x	x	1	1	0	RTR	x	x	x	x

Tab. 3 Příklad nastavení filtru zpráv

V předešlé tabulce je uveden příklad nastavení filtru zpráv, kde pouze identifikátor ID bude přijat (x znamená, že na dané hodnotě nezáleží).

- **void** InitInterrupt **(void)** – inicializace přerušení od řadiče CAN sběrnice

Na straně řadiče stačí k inicializaci přerušení nastavit bit RIE (*Receive Interrupt Enable*) povolující přerušení po příjmu zprávy na log.1, který se nachází v registru povolení přerušení Interrupt Enable (adresa 4). Jeho hodnota bude tedy 0x01.

Na straně mikrokontroléru je třeba nastavit povolení externího přerušení. K tomuto účelu jsou zde registry GIMSK a MCUCR. Pomocí registru GIMSK se

povoluje či zakazuje přerušení od vstupu INT0 (6.bit) resp. INT1 (7.bit). Jelikož je vnější přerušení přivedeno na vstup INT0, nastaví se registr GIMSK na hodnotu 0x40. Pomocí bitů registru MCUCR lze konfigurovat podmínku, která aktivuje přerušení vnějším vstupem. Bity 2 (ISC10) a 3 (ISC11) se týkají vstupu INT1 a bity 0 (ISC00) a 1 (ISC01) vstupu INT0. Význam těchto bitů zobrazuje následující tabulka.

ISC01	ISC00	popis
0	0	vstup INT0 se aktivuje lo.0 (úrovňově citlivý vstup)
0	1	rezervováno
1	0	vstup INT0 se aktivuje sestupnou hranou (hranově citlivý vstup)
1	1	vstup INT0 se aktivuje náběžnou hranou (hranově citlivý vstup)

Tab. 4 Příklad nastavení filtru zpráv

Příjem a vysílání zpráv po CAN sběrnici

- **void** ReceiveMessage **(void)** – načtení přijaté zprávy do pole hodnot *rbuffer[]* a nastavení příznaku vydání přijaté zprávy RRB (*Release Receive Buffer*) v registru *command* (0x04)

- **void** TransmiteMessage **(void)** – uložení zprávy pro vyslání z pole hodnot *sbuffer[]* na příslušná místa v paměťovém prostoru řadiče SJA1000 a nastavení příznaku žádosti o vyslání TR (*Transmition Request*) v registru *command* (0x01)

- **void** SendMsg **(unsigned int** NodeID**, char** DLC**, unsigned char** data[8]**)** – naplnění pole *sbuffer[]* hodnotou udávající počet vysílaných datových bytů *DLC*, identifikátorem zprávy *NodeID* (11ti bitový) a příslušným počtem datových bytů *data[]*. Následně se zavolá funkce *TransmiteMessage()*.

- **void** SendMsgEXT **(unsigned int** NodeIDH**, unsigned int** NodeIDL, **char** DLC**, unsigned char** data[8]**)** – obdobná funkce jako funkce *SendMsg*, pouze s tím rozdílem, že se jedná o rozšířený identifikátor zprávy (29ti bitový).

Rutiny přerušení

V této části jsou popsány činnosti přerušení vyvolané přijetím znaku po sériové lince, přetečením časovače 1 a od vnějšího vstupu přerušení na vývodu INT0.

- **interrupt** [UART_RXC] **void** uart_int **(void)** – obsluha přerušení po příjmu znaku od UARTu. Přijatý znak se uloží do proměnné *tlacitko*, na základě jehož hodnoty se v hlavním programu provede příslušná odezva. Přijde-li však znak ‘s‘, znamená to, že bude následovat série 16ti hodnot pro nastavení parametrů komunikace po CAN sběrnici. Tyto hodnoty se postupně uloží do řetězce *str* pro jejich následné zpracování. Po uložení všech hodnot se nastaví příznak *precteno*.

- **interrupt** [TIM1_OVF] **void** timer1_ovf_isr **(void)** – tvorba hodinových impulsů pro ovládání krokových motorků. Na základě hodnot v registrech *RegH* a *RegL* se volí frekvence výstupních impulsů (viz. 3.1.1). Dále se zde čítá počet kroků, po jejichž ujetí se nastaví příznak *Konec*.

- **interrupt** [EXT_INT0] **void** ext_int0 **(void)** – obsluha přerušení od vnějšího vstupu INT0, signalizující přijetí datové zprávy CAN řadičem SJA1000. Pomocí funkce *ReceiveMessage()* se provede uložení této zprávy, následně se zjistí její formát (standardní či rozšířený) a nastaví se příznak jejího přijetí *msgread* na jedna.

3.1.3 Popis hlavní části programu

Hlavní část program se odehrává v nekonečné smyčce, které předchází přednastavení důležitých registrů a inicializace CAN sběrnice. V této smyčce se vyhodnocují odezvy na nastavení příslušných příznaků. Těmito činnostmi jsou uložení nastavení z PC do EEPROM a přečtení tohoto nastavení z EEPROM do PC, přijímání a vysílání zpráv po CAN sběrnici, a reakce na stisk jednotlivých tlačítek, ať už přímo na desce, nebo v PC.

Uložení a nastavení parametrů z PC

V případě, že je nastaven příznak *precteno*, dojde k uložení hodnot v řetězci *str* (pole znaků char) do příslušných proměnných umístěných v paměti EEPROM. Těmito

hodnotami jsou, přenosová rychlost (*baudrate*), výstupní mód (*outmode*), registr výstupního hodinového signálu (CDR), registr pro filtraci příchozích zpráv (ACR, AMR) a identifikátor zpráv. Podle toho jaká je hodnota v proměnné *format* se ukládá buď standardní (*format* = 0x00) nebo rozšířený (*format* = 0xFF) identifikátor zprávy. Po uložení těchto hodnot v EEPROM se provede inicializace nastavení komunikace po CAN sběrnici zavoláním funkce *InitCAN()*.

Přečtení a vyslání aktuálního nastavení do PC

Přečtení aktuálního nastavení z paměti EEPROM do PC, je iniciováno posláním znaku ‘c’ z PC po sériové lince RS232. V případě, že je tedy hodnota proměnné *tlacitko* rovna právě znaku ‘c‘, tak se provede načtení hodnot proměnných z EEPROM do řetězce *str1*. Opět se dbá na to, zda se jedná o standardní, či rozšířený identifikátor zprávy. Po načtení všech hodnot se provede odvysílání tohoto řetězce do PC.

Přijetí a vyhodnocení zprávy přijaté po CAN

Jakmile přijde zpráva, vyvolá se přerušení, zpráva se uloží do *rbuffer[]* a nastaví se příznak *msgread*. V případě, že je tento příznak nastaven, dojde k jejímu vyhodnocení. Formát této zprávy záleží opět na tom, zda se jedná o standardní či rozšířený identifikátor. Struktura této zprávy je znázorněna v následující tabulce.

CAN adresa	název registru	význam obsahu
16	frame information	počet bajtů
17	identifier 1	identifikátor
18	identifier 2	identifikátor
19	data byte 1	směr + stop
20	data byte 2	počet kroků
21	data byte 3	
22	data byte 4	rychlost
23	data byte 5	
24	data byte 6	nevyužito
25	data byte 7	nevyužito
26	data byte 8	nevyužito
27	unused	nevyužito
28	unused	nevyužito

CAN adresa	název registru	význam obsahu
16	frame information	počet bajtů
17	identifier 1	identifikátor
18	identifier 2	identifikátor
19	identifier 3	identifikátor
20	identifier 4	identifikátor
21	data byte 1	směr + stop
22	data byte 2	počet kroků
23	data byte 3	
24	data byte 4	rychlost
25	data byte 5	
26	data byte 6	nevyužito
27	data byte 7	nevyužito
28	data byte 8	nevyužito

Tab. 5 Formát datové zprávy – standardní a rozšířený formát

Každá zpráva obsahuje pět bytů určujících činnosti krokového motorku. První byte spouští motorek v daném směru, nebo ho zastavuje. Možné hodnoty jsou patrné z následující tabulky.

hodnota	činnost
0x00	stop
0x0F	vpravo
0xF0	vlevo

Tab. 6 Obsah prvního datového bytu

Pokud je hodnota rovna 0x00, vyresetuje se motorek (PORTC.0 = 1) a zakáže se možnost přerušení od čítače/časovače 1, čímž se motorek zastaví. V opačných případech se povolí přerušení a nastaví se příslušný směr otáčení (PORTC.1) na log.0 (vpravo) nebo log.1 (vlevo).

Další dva byty tvoří 16ti bitovou hodnotu (*Steps*) udávající počet kroků, které má motorek ujet. V případě, že je tato hodnota rovna nule, tak se motorek rozjede a nepočítá se počet kroků. Poslední dvojice bytů ovládá rychlost otáčení motorku. Tyto hodnoty se uloží do proměnných *RegH* a *RegL* pro nastavení periody výstupního hodinového signálu na vývodu PORTC.2, jehož hodnota se mění v rutině přerušení čítače/časovače 1.

Vysílání zprávy po CAN

Po ujetí požadovaného počtu kroků se nastaví příznak *Konec*, na jehož základě se vyšle datová zpráva, kterou tvoří dva byty naplněné hodnotou 0xFF. Dle formátu zprávy se zavolá buď funkce *SendMsg()* pro standardní formát, nebo *SendMsgEXT()* pro rozšířený formát identifikátoru datové zprávy.

Ovládání pomocí tlačítek

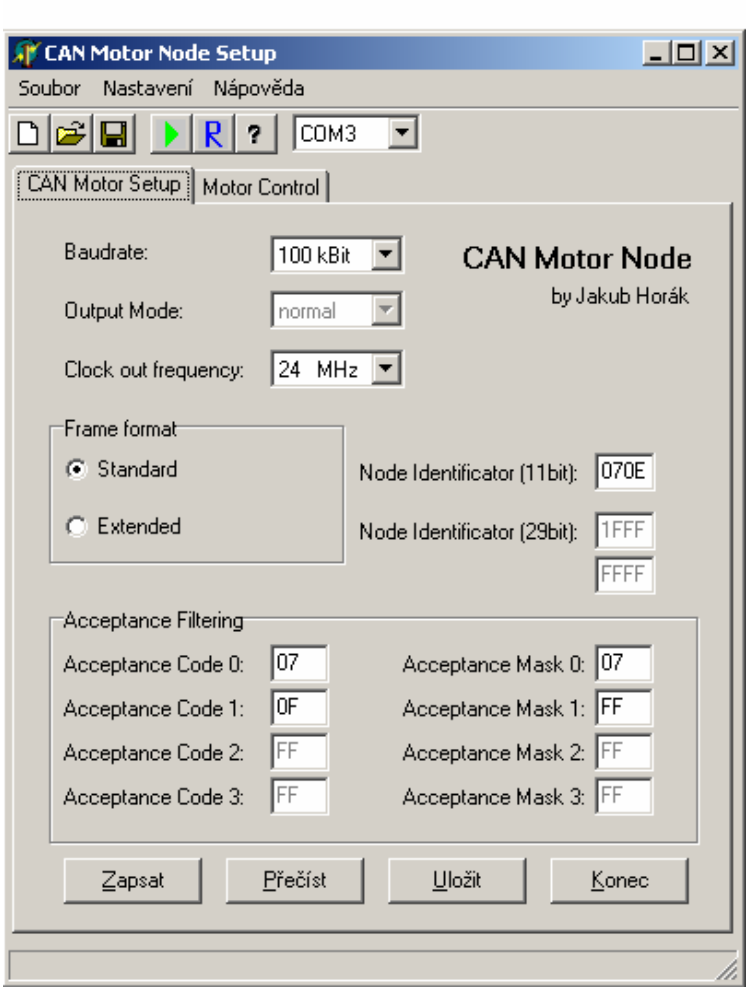
Realizovaný uzel obsahuje také trojici tlačítek, jimiž je možno přímo ovládat krokový motorek. Jednotlivá tlačítka mají funkce jízdy vlevo, vpravo a zastavení motorku. Stejné činnosti lze dosáhnout i ovládáním motorku z aplikace vytvořené v PC. Ovládajícími povely jsou přijaté znaky ‘l’ pro jízdu vlevo, ‘r’ pro jízdu vpravo a ‘z’ pro zastavení motorku.

3.2 Nastavování vlastností uzlu CAN z PC

Nastavování potřebných vlastností ke správné komunikaci realizovaného uzlu po sběrnici CAN se provádí z PC po sériové lince RS232. K tomuto účelu slouží aplikace s názvem *CANMotorSetup*, která byla vytvořena ve vývojovém prostředí Delphi 6.0. Tato aplikace umožňuje zadávat potřebné parametry do editačních polí a zakódované je poslat, prostřednictvím sériové linky, do mikrokontroléru uzlu CAN, kde jsou nejprve uloženy v paměti EEPROM a následně předány řadiči sběrnice CAN. Aplikace dále umožňuje tyto hodnoty také přečíst a následně uložit do souboru pro jejich pozdější využití. Kompletní zdrojový kód je možno nalézt na přiloženém CD.

3.2.1 Vzhled a ovládání aplikace

Na následujícím obrázku je zobrazen vzhled vytvořené aplikace.



Obr. 26 Vzhled aplikace CAN Motor Node

Pro srozumitelnost nastavování jednotlivých položek, jsou jejich názvy vybaveny plovoucími nápovědami a zároveň je jejich popis zobraz i na stavovém řádku aplikace. Dále ja aplikace vybavena nápovědou, ve které jsou podrobněji vysvětleny možnosti nastavování.

Ovládání aplikace je možné buď pomocí roletového menu, panelu nástrojů, případně přímo pomocí tlačítek. Jednotlivé funkce jsou totožné, tzn. že jestliže se zvolí např. funkce *Uložit* v nabídce roletového menu, či přímo tlačítko na ploše nebo v panelu nástrojů, vždy se provede stejná činnost.

Roletové menu se skládá ze tří položek jimiž jsou *Soubor*, *Nastavení* a *Nápověda*. Roleta *Soubor* (Alt+"S") obsahuje položky *Nový*, *Otevřít*, *Uložit* a *Konec*. Volbou *Nový* se všechny editační pole vymažou, resp. nastaví se na hodnoty nula. Kliknutím na položkou *Otevřít* se otevře dialogové okno pro otevření textového souboru s uloženou konfigurací. Obdobně volba *Uložit* otevře dialog pro uložení daného nastavení do textového souboru. Volbou *Konec* se ukončí činnost aplikace.

Roleta Nastavení (Alt+"A") obsahuje položky *Zapsat*, *Přečíst* a *Výchozí*. Volbou *Zapsat* se provede zapsání této konfigurace do mikrokontroléru uzlu CAN. Položkou *Přečíst* lze získat jeho aktuální nastavení. Výchozí nastavení se získá zvolením stejnojmenné položky.

Poslední volbou je položka *Nápověda* (Alt+"N"), po jejímž stisku se otevře formulář s textovou nápovědou vysvětlující základní ovládání této aplikace.

Jednotlivé volby jsou rovněž zobrazeny na panelu nástrojů. Jsou tematicky odděleny separátory podle nabídek roletového menu. Zleva jsou položky *Nový*, *Otevřít* a *Uložit*, následuje separátor, po němž jsou volby *Zapsat*, *Přečíst* a *Nápověda*. Panel nástrojů rovněž obsahuje volbu komunikačního rozhraní. Je zde možnost volby ze čtyř sériových portů (COM1 až COM4). Pro pohodlnější a rychlejší obsluhu jsou nejdůležitější funkce přístupné přímo z formuláře pomocí tlačítek *Zapsat* (Alt+"Z"), *Přečíst* (Alt+"P"), *Uložit* (Alt+"U") a *Konec* (Alt+"K").

Vlastní vzhled aplikace tvoří dvě záložky, na první z nich s názvem *Nastavení uzlu CAN* jsou umístěna veškerá potřebná zadávací pole pro nastavení správné činnosti uzlu CAN. Druhá z nich – *Ovládání motorku*, je pouze doplňková s funkcí přímého ovládání krokového motorku z PC (pouze jízda v určitém směru a zastavení). Jednotlivé zadávací položky mají vedle sebe popisek, který odpovídá názvu příslušného registru

řadiče CAN. V případě nutnosti bližší informace o zadávané položce, se po najetí na ni objeví plovoucí nápověda v češtině a zároveň se její text zobrazí i na stavovém řádku.

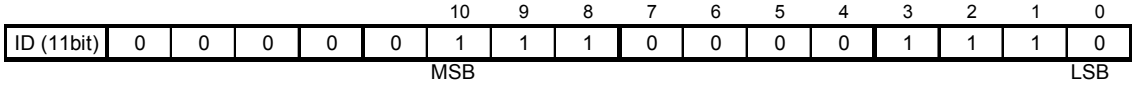
První volbou je volba přenosové rychlosti z rozbalovacího boxu s názvem *Baudrate*. Zde je možnost konkrétní volby hodnoty přenosové rychlosti. Do této položky není možnost zadávat vlastní hodnoty, lze vybrat pouze z následujících předvolených hodnot: 10kBit, 20kBit, 50kBit, 100kBit, 125kBit, 250kBit, 500kBit, 800kBit a 1Mbit. Výhodou tohoto řešení je, že uživatel nepotřebuje zadávat konkrétní hodnoty do příslušných registrů (viz.3.1.2), neboť to se provede až na základě této volby přímo v mikrokontroléru uzlu. Další nabídkou je volba výstupního módu, tato položka není ovšem přístupná, je zde ponechána pouze pro informaci uživatele v jakém módu je zapojen řadič CAN sběrnice. Následuje volba výstupní frekvence na vývodu CLKOUT řadiče SJA1000. Tento řadič dává možnost vytvořit signál určité frekvence na jednom ze svých vývodů. Ačkoli není v současné verzi uzlu vývod CLKOUT využit, je volba této frekvence ponechána pro její možné budoucí využití. Hodnoty jednotlivých výstupních frekvencí jsou dány podíly frekvence připojeného krystalu f_{OSC} (f_{OSC} = 24MHz):

$$f_{OSC}, \frac{f_{OSC}}{2}, \frac{f_{OSC}}{4}, \frac{f_{OSC}}{6}, \frac{f_{OSC}}{8}, \frac{f_{OSC}}{10}, \frac{f_{OSC}}{12}, \frac{f_{OSC}}{14},$$

konkrétní možné hodnoty tedy jsou (v MHz): 24; 12; 6; 4; 3; 2,4; 2 a 1,7 MHz.

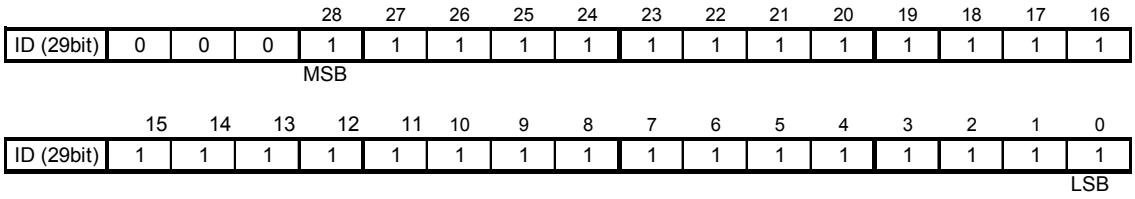
Velice důležitými položkami jsou možnosti zadávání identifikátorů vysílacích zpráv a filtru příchozích zpráv. Nejprve je zde volba formátu datového rámce, zda se jedná o standardní či rozšířený identifikátor zprávy. V případě že je zvolen standardní formát, tak se povolí pouze jemu příslušné položky a položky pro nastavení rozšířeného formátu se zakážou. Stejně tak to platí i v druhém případě. Zadávání konkrétních hodnot do jednotlivých políček je v hexadecimálním formátu. Z tohoto důvodu jsou jednotlivá zadávací pole ošetřena proti zápisu nežádoucích hodnot, povoleny jsou pouze alfanumerické hodnoty “0“ až “9“ a “a“ až “f“ nebo “A“ až “F“. Nezáleží na tom zda se napíší malá nebo velká písmena, neboť malá jsou následně převedena na velká.

Standardní formát identifikátoru osahuje 11 bitů, proto prvních pět bitů je nastaveno na nulu a poté následuje identifikátor. Jeho maximální hodnota může být tedy 0x07FF. Přednastavená hodnota je 0x070E, znázornění je na následujícím obrázku.



Obr. 27 Formát standardního identifikátoru zprávy – 11bitů

Rozšířený identifikátor zprávy obsahuje 29 bitů, první tři bity jsou proto nastaveny na nulu. Maximální možnou hodnotou je hodnota 0x1FFF pro horní polovinu a 0xFFFF pro dolní, viz. následující obrázek.



Obr. 28 Formát rozšířeného identifikátoru zprávy – 29 bitů

V případě, že je zadána hodnota identifikátoru vyšší než maximální možná hodnota a přesto se provede žádost o zapsání nastavení do paměti mikrokontroléru uzlu, tak se vygeneruje varování o nesprávně zadaném identifikátoru.

Pro zadávání filtru příchozích zpráv je třeba nastavit dva druhy registrů. Jsou jimi *Acceptance Code* (ACR1 až ACR4) a *Acceptance Mask* (AMR1 až AMR4), jejich význam je popsán v kapitole 3.1.2 v tabulce Tab.3. Registr ACR slouží pro zadání identifikátoru přijímané zprávy a registr AMR pro masku této zprávy. Zadávání hodnot do těchto políček je ošetřeno obdobně jako u identifikátoru vysílající zprávy. Před nahráním těchto hodnot do mikrokontroléru uzlu je nejprve nutno provézt posun o pět bitů (standardní identifikátor), resp. o tři bity vlevo (rozšířený). Toto je provedeno z důvodu přehlednosti zadávání konkrétních hodnot identifikátorů, kdy se zadá přímo jeho správná hodnota bez ohledu na vlastní interpretaci v příslušných registrech řadiče.

3.2.2 Zápis a čtení nastavení uzlu CAN

Komunikace mezi PC a CAN uzlem probíhá po sériové lince RS232. Po spuštění aplikace se nastaví její parametry. Komunikace probíhá po zvoleném sériovém portu s přenosovou rychlostí 9600 Baudů. Přenášená zpráva obsahuje jeden start bit, 8 datových bitů, žádný paritní bit a jeden stop bit.

Zápis zadaného nastavení se provede stiskem tlačítka *Zapsat* na formuláři, či v příslušné nabídce roletového menu, nebo na panelu nástrojů. Po jeho stisku se zobrazí potvrzovací okno s možností potvrzení či zrušení této akce. V případě jeho potvrzení se již provede vlastní zápis hodnot do mikrokontroléru uzlu. Tyto hodnoty jsou zadávány v hexadecimálním formátu a pro vlastní přenos je nutné převést je na nějaký vhodný formát. Pomocí vytvořených procedury *Hex2Int* (1 byte) nebo *Transfer* (2 byty) se hexadecimální hodnoty z editačních polí převedou na jejich dekadická vyjádření (př. 0xFF odpovídá hodnotě 255). Tyto hodnoty se poté převedou pomocí procedury *Byte2Str* na odpovídající znaky ASCII tabulky, a postupně se ukládají do řetězcové proměnné. Aby mikrokontrolér poznal že se jedná o data určená pro nastavení nové konfigurace, je na začátek tohoto řetězce vložen znak "s". Poté již dojde k vlastnímu vyslání tohoto řetězce znak po znaku do mikrokontroléru uzlu. Pro rozlišení zda se jedná o standardní či rozšířený formát identifikátoru zprávy se před něj vloží buď hodnota 0x00 (standardní) nebo hodnota 0xFF (rozšířený). V případě standardního formátu se řetězec doplní hodnotami 0x00 pro konstantní délku 17ti znaků (včetně znaku "s"). Formát vysílaného řetězce a jeho možných hodnot (před jejich upravením a převedením) je zobrazen v následující tabulce.

pozice	název proměnné	možné hodnoty
0	znak "s"	"s"
1	Baudrate	0x00 - 0x08
2	Output mode	0x1A
3	Clock out frequency	0x00 - 0x07
4	Acceptance Code 0	0x00 - 0x07
5	Acceptance Code 1	0x00 - 0xFF
6	Acceptance Code 2	0xFF
7	Acceptance Code 3	0xFF
8	Acceptance Mask 0	0x00 - 0x07
9	Acceptance Mask 1	0x00 - 0xFF
10	Acceptance Mask 2	0xFF
11	Acceptance Mask 3	0xFF
12	formát zprávy	0x00
13	Node Identificator 0	0x00 - 0x07
14	Node Identificator 1	0x00 - 0xFF
15		0x00
16		0x00

pozice	název proměnné	možné hodnoty
0	znak "s"	"s"
1	Baudrate	0x00 - 0x08
2	Output mode	0x1A
3	Clock out frequency	0x00 - 0x07
4	Acceptance Code 0	0x00 - 0x1F
5	Acceptance Code 1	0x00 - 0xFF
6	Acceptance Code 2	0x00 - 0xFF
7	Acceptance Code 3	0x00 - 0xFF
8	Acceptance Mask 0	0x00 - 0x1F
9	Acceptance Mask 1	0x00 - 0xFF
10	Acceptance Mask 2	0x00 - 0xFF
11	Acceptance Mask 3	0x00 - 0xFF
12	formát zprávy	0xFF
13	Node Identificator 0	0x00 - 0x1F
14	Node Identificator 1	0x00 - 0xFF
15	Node Identificator 2	0x00 - 0xFF
16	Node Identificator 3	0x00 - 0xFF

Tab. 7 Formát vysílaného řetězce a rozsah jeho možných hodnot – standardní a rozšířený formát

Jestliže uživatel vyžaduje zjištění aktuálního nastavení uzlu, docílí toho stiskem tlačítka *Přečíst* na formuláři, nebo v příslušné nabídce roletového,či opět na panelu nástrojů. Po jeho stisku je vyslána žádost o přečtení nastavení do

mikrokontroléru. Tuto žádost reprezentuje vyslání znaku “c“. Mikrokontrolér tento znak vyhodnotí a na jeho základě pošle sadu hodnot aktuálního nastavení uloženou ve vnitřní paměti EEPROM. Aplikace tyto hodnoty přijímá postupně v dekadické formě reprezentující příslušný ASCII znak. Tyto hodnoty jsou zpětně převedeny na hexadecimální vyjádření funkcí *Char2Hex* a jsou zobrazeny v příslušných editačních polích.

3.2.3 Ukládání konfigurace do souboru

Funkci ukládání konfigurace do souboru je možno zvolit opět několika způsoby, buď tlačítkem *Uložit* umístěném na formuláři, případně ikonou na panelu nástrojů, nebo příslušnou volbou v nabídce roletového menu *Soubor*. Po volbě jedné z nich se otevře dialogové okno pro uložení nastavení do souboru. Je zde nastaven filtr pouze pro textové soubory s příponou *.txt. Každá hodnota z editačního pole je ukládána na samostatný řádek v souboru. Formát ukládaných dat je obdobný jako při zápisu do paměti mikrokontroléru (viz. Tab. 7), neukládá se však znak “s“.

Načtení hodnot z textového souboru se uskuteční po stisku tlačítka na panelu nástrojů, případně v nabídce *Soubor – Otevřít*. Otevře se dialogové okno pro otevření souboru dle nastaveného filtru povolujícího pouze soubory textové s příponou *.txt. Poté dojde k zapsání těchto hodnot do příslušných editačních polí a nastavení přepínače formátu identifikátoru datové zprávy.

3.3 Ověření realizace pomocí vývojového systému s mikropočítačem C167CR

3.3.1 Stručný popis vývojového systému

Vývojový systém, jehož jádrem je mikroprocesor firmy Infineon C167CR-LM, je výrobkem firmy Hitex Ltd. Tento mikroprocesor je součástí desky C167IO, která je dále připojena k desce uživatelských vstupů a výstupů ERTEC EVA167. Napájení desky je 12VDC. Tato deska poskytuje uživateli dvouřádkový šestnáctiznakový LCD displej Hitachi, maticovou klávesnici s šestnácti tlačítky, pět potenciometrů, deset LED diod, osm DIL přepínačů pro volbu adresového prostoru

paměti, vstupní a výstupní analogový filtr, proudové výstupy MOSFET, signály reálného času C515C a hlavně budič sběrnice CAN – již zmiňovaný obvod Philips PCA82C250. Řadič sběrnice CAN je součástí mikroprocesoru C167CR-LM, který obsahuje až 15 objektů zpráv.

3.3.2 Popis programu pro C167CR

Mikroprocesor C167CR je možno programovat buď v assembleru, lépe však v programovacím jazyku C. Program je tedy napsán v jazyce C ve vývojovém prostředí µVision3 od firmy Keil Software. Následná simulace činnosti programu je provedena pomocí programu HITOP firmy Hitex Ltd. Zdrojový kód programu je možno nalézt na přiloženém CD.

Nejprve je nutno nastavit parametry komunikace po CAN sběrnici. Toto nastavení je provedeno v modulu *can.c*, kde je vytvořena funkce *initialise_CAN ()*. Touto funkcí je určena přenosová rychlost komunikace 100kBit/s a vzorkovací bod 75%, nastavením registru časování bitu na hodnoty BRP = 4, TSEG1 = 13, TSEG2 = 4 a SJW = 3. Dále jsou zde inicializovány objekty zpráv jako vysílací či přijímací a jsou jim přiřazeny identifikátory. Objekt zprávy 0 (*CAN_Object[0]*) je nastaven jako přijímací (2 byty) s identifikátorem zprávy 0x70E a objekt zprávy 1 (*CAN_Object[1]*) jako vysílací (5 bytů) s identifikátorem 0x70F.

Objekt zprávy	Identifikátor	Směr	Počet bajtů	Funkce
0	0x70E	přijímací	2	Signalizace ujetí zadaného počtu kroků (2B)
1	0x70F	vysílací	5	Směr (1B)+Počet kroků (2B)+Rychlost (2B)

Tab. 8 Nastavení parametrů objektů zpráv

Hlavní činnost programu se odehrává v modulu *main.c*. Vysílání hodnot po sběrnici CAN určených pro realizovaný uzel se provádí cyklicky s periodou 200 ms. V této rutině přerušení (*T1_interrupt()*) od časovače T1 se zjišťuje stav potenciometrů pro zadávání počtu kroků a rychlosti otáčení krokového motorku. Tyto stavy se vloží na příslušná místa datové zprávy a vyšlou se na CAN sběrnici.

Jednotlivé činnosti krokového motorku se ovládají z maticové klávesnice, jejíž stav se neustále zjišťuje v nekonečné smyčce hlavního programu. Rozjetí motorku doprava (po směru hodinových ručiček) se spustí stiskem klávesy “3“, doleva (proti směru hodinových ručiček) klávesou “1“. V průběhu jízdy motorku lze jeho rychlost měnit potenciometrem P2. Hodnota natočení potenciometru se zobrazuje na LED diodách. Motorek se zastaví stiskem klávesy “5“. Další možností je zadání počtu kroků, které má motorek ujet. To se provádí potenciometrem P3, jehož hodnota se rovněž zobrazuje na LED diodách. Jakmile se poté stiskne klávesa “6“, tak se motorek rozjede doprava, v případě klávesy “4“ doleva. V nekonečné smyčce se dále zjišťuje zda nepřišla zpráva od CAN uzlu sdělující, že již bylo provedeno ujetí zadaného počtu kroků. To je signalizováno přijetím hodnoty 0xFFFF. Poté se krokový motorek zastaví vysláním hodnoty 0x00 v prvním bytu datové zprávy.

3.3.3 Vyhodnocení návrhu

Během ověřování činnosti komunikace mezi tímto vývojovým systémem a realizovaným uzlem CAN nebyly zjištěny žádné problémy. Uzel správně reagoval na vysílané požadavky podle nichž ovládal krokový motorek. Ověřena byla zároveň i činnost nastavování parametrů komunikace z PC prostřednictvím sériové linky RS232. Zadáním nesprávného identifikátoru vysílajících zpráv z uzlu CAN nedocházelo k přijímání žádostí na zastavení motorku po ujetí žádaného počtu kroků, proto se motorek nezastavil, což odpovídalo předpokladům. Vyzkoušena byla také funkce filtrování příchozích zpráv, kdy nastavení hodnoty filtru na 0x0000 bralo v potaz konkrétní identifikátor, zatímco při hodnotě 0x07FF nezáleželo na jeho nastavení.

3.4 Možnosti programování v aplikační vrstvě

3.4.1 Základní informace o aplikační vrstvě

Protokol CAN obsahuje popis pouze fyzické a spojové vrstvy podle modelu ISO/OSI. Popsané služby zajišťují vysílání a příjem zpráv spolu s poměrně mocnými zabezpečovacími mechanismy. Takto realizované systémy jsou obvykle šité na míru konkrétní aplikaci. To příliš nevadí při použití v automobilech, problematičtější je však

použití CAN jako průmyslové řídící sběrnice. Je proto třeba, aby protokol definoval i vrstvu aplikační, která dodává význam datům přenášeným vrstvou spojovou. Těchto protokolů existuje více. Nejširší uplatnění si získal protokol aplikační vrstvy CAL (CAN Application Layer) standardizovaný zejména pro použití v průmyslové automatizaci. Standard CAN poskytuje čtyři oblasti služeb: *CAN Based Message Specification* (CMS), *Network Management* (NMT), *Distributor* (DBT) a *Layer Management* (LMT).

CMS definuje základní komunikační služby a objekty pro jednoduchý návrh distribuovaného systému. Charakteristickým znakem pro jazyk CMS je architektura klient-server. Serverem je zařízení obsahující CMS objekty s nimiž mohou pracovat CMS klienti. Dalším charakteristickým znakem je objektově orientovaný návrh spočívající v použití objektů zapouzdřujících atributy. Specifikace dále definuje metody, pomocí kterých se k objektům přistupuje. Ty lze rozdělit na lokální, přistupují pouze k objektům umístěným na lokálním zařízení, a vzdálené, přistupujících k objektům na vzdálených zařízeních. Jsou rozeznávány tři základní datové typy komunikačních objektů: proměnné, domény a události. **Proměnné** (*Variables*) reprezentují reálná data v zařízení. Datové typy proměnných mohou být jednoduché (boolean, integer, float atd.), složené (array, struct), nebo odvozené ze složených. Jejich délka je do osmi byte. Pro objekty typu proměnná jsou definovány služby zápisu a čtení dat do resp. z objektu. **Domény** (*Domains*) jsou objekty reprezentující blok dat (libovolné délky) nespecifikovaného typu, který je pro přenos rozdělen do několika telegramů. Používají se zejména pro přenos konfiguračních dat, programů apod. **Události** (*Events*) jsou narozdíl od proměnných a domén iniciovány serverem. Chovají se tedy jako přerušení, které je buď řízené nebo neřízené.

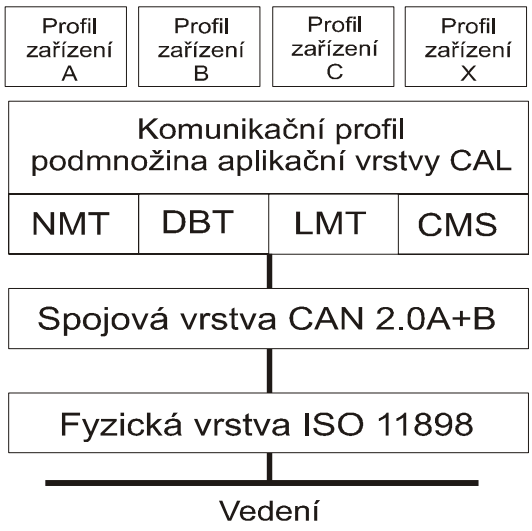
Cílem NMT služeb je poskytnout nástroje pro vytváření distribuovaných systémů a jejich údržbu. Používá se komunikační model master-slave, který je zcela nezávislý na komunikaci prostřednictvím CMS. Řídící stanice (master) zabezpečuje konfigurování sítě a během provozu kontroluje správnou funkci ostatních stanic.

NMT těsně spolupracuje s DBT, která je částí aplikační vrstvy mající na starost přidělování identifikátorů komunikačním objektům vytvořených CMS a jejich správu. Při jejich přidělování je nutné zamezit přidělení různých identifikátorů stejnému komunikačnímu objektu či stejného identifikátoru různým komunikačním objektům.

LMT služby umožňují měnit některé základní parametry zařízení připojených ke sběrnici, především přenosovou rychlost.

3.4.2 Komunikační model CANopen

Protokol aplikační vrstvy CANopen rozšiřuje některé služby a protokoly definované vrstvou CAL a navíc definuje množství profilů pro většinu běžných zařízení používaných v průmyslové automatizaci. Tento protokol je určen pro řídící aplikace na nižší systémové úrovni především s ohledem na potřeby průmyslového řízení v reálném čase. Struktura protokolu CANopen je znázorněna na obr.29. Fyzická a spojová vrstva jsou realizovány pomocí sběrnice CAN. Podmnožina aplikační vrstvy CAL definuje činnost sítě a jednotlivých zařízení z hlediska komunikace. Dále jsou vytvořeny profily jednotlivých zařízení definující určité standardní funkce a další jejich vlastnosti. Důraz je přitom kladen na to, aby bylo možno snadno zaměňovat zařízení obdobného druhu od různých výrobců. Tímto způsobem jsou definovány profily pro analogové i číslicové vstupy a výstupy, programovatelné automaty, pohony a další kategorie zařízení.



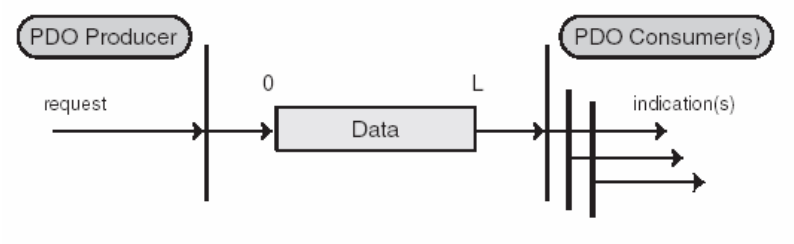
Obr. 29 Struktura protokolu CANopen dle referenčního modelu ISO/OSI viz.[9]

Popis funkčnosti zařízení, parametrů, datových typů apod. se nachází v tzv. adresáři objektů (*Object Dictionary*), který obsahuje část s obecnou specifikací zařízení (název, výrobce, komunikační parametry atd.) a část obsahující určitou funkci zařízení, parametry a data. Jednotlivé položky v adresáři se nazývají objekty.

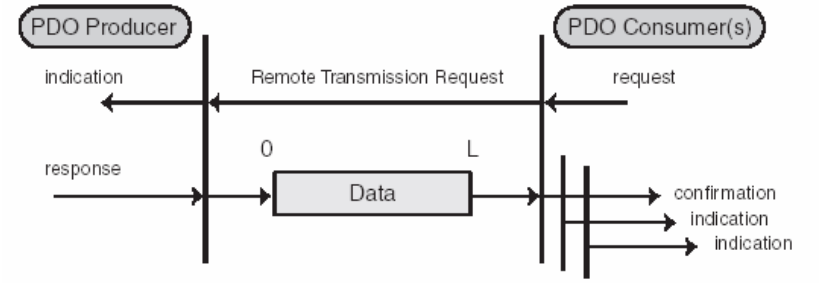
Na protokolu CANopen se rozlišují dva typy mechanismů přenášení zpráv, kterými jsou procesní data PDO (*Process Data Objects*) a služební zprávy SDO (*Service Data Objects*). Procesní data určená pro časově kritickou výměnu se přenášejí buď cyklicky, při změně nebo na vyžádání bez dodatečné režie. Jedná se o

nepotvrzovaný přenos datových rámců. Druhým mechanismem jsou služební zprávy, jejichž doručení není tolik omezeno časem. Tyto zprávy se používají především pro přenášení parametrů při konfiguraci zařízení nebo pro přenášení delších zpráv.Tento způsob přenosu vyžaduje dodatečnou režii. Jedná se o potvrzovaný přenos.

Přenos procesních dat PDO probíhá nepotvrzovanými zprávami podle modelu znázorněného na obrázcích obr.30 pro zápis dat a obr.31 pro čtení dat na vyžádání.



Obr. 30 Přenos procesních dat PDO – zápis viz.[13]



Obr. 31 Přenos procesních dat PDO na vyžádání – čtení viz.[13]

Každému PDO je přiřazen jeden konkrétní identifikátor určující význam přenášených dat. Obrázek obr.30 ukazuje zápis procesních dat, tedy přenos od zařízení *PDO-producer* (producent) k jednomu nebo více zařízením *PDO-consumer* (konzument). Producentem dat je typicky zařízení – čidlo, které hodnoty svých vstupů přenáší na sběrnici, kde následně tato data přijme Konzument. Obrázek obr.31 zobrazuje vyslání žádosti o data konzumentem a následné vyslání požadovaných dat producentem pro možné přečtení všemi konzumenty.

U přenosu dat SDO se jedná o potvrzovaný přenos dat libovolné délky podle modelu klient-server. Pro každou zprávu v příslušném směru je třeba jeden identifikátor zprávy CAN. Spojení je vytvořeno vždy mezi dvěma účastníky. Přenos libovolné délky je možný díky rozdělení zprávy na jednotlivé segmenty s maximální délkou 8 bytů.

Závěr

Cílem této diplomové práce byl návrh a realizace uzlu s CAN protokolem pro připojení krokového motorku. Nastavování parametrů komunikace po CAN sběrnici mělo být umožněno z PC po sériové lince.

Použité komponenty pro vlastní realizaci uzlu byly vybírány zejména z důvodu zabezpečení funkčnosti a jednoduchosti celého zařízení, neméně také z důvodu jejich ceny a dostupnosti. Uzel je realizován na univerzální desce plošných spojů, rovněž je navržena deska pomocí návrhového prostředí Eagle. V případě použití tohoto uzlu v nějaké konkrétní aplikaci je možno využít SMD součástek a docílit tak mnohem menších rozměrů desky, pro její lepší aplikovatelnost.

Takto realizovaný uzel umožňuje klasické možnosti ovládání krokových motorků, jako je změna směru otáčení, změna rychlosti a jelikož se jedná o motorek krokový, tak i zadávání počtu kroků. Vzhledem k tomu, že ovládání činnosti krokového motorku nevyžaduje příliš mnoho parametrů, vystačí se pouze s jedním typem datového rámce obsahujícím informaci o směru (1byte), počtu kroků (2byty) a rychlosti otáčení (2byty). Z tohoto důvodu mají datové zprávy vždy konstantní délku. Výhodou je jednoduchost komunikace vyžadující pouze znalost jednoho identifikátoru pro uzel CAN. Avšak má to i svou drobnou nevýhodu spočívající v rychlosti přenášeného datového rámce, vzhledem k jeho konstantní délce, která ovšem není příliš rozhodující díky poměrně vysokým komunikačním rychlostem protokolu CAN. Toto by se dalo vyřešit více typy datových rámců lišících se svými identifikátory, což by ovšem vedlo na složitější strukturu řídícího programu.

Pro zadávání parametrů komunikace z PC prostřednictvím sériové linky RS232 je navržena aplikace ve vývojovém prostředí Delphi. Její vzhled je nápadný svojí jednoduchostí a srozumitelností. Jednotlivé parametry, jako je přenosová rychlost komunikace, jsou zadávány přímo jejich číselnou hodnotou, převod na konkrétní hodnoty nastavení potřebné pro řadič CAN se provede před vlastním vysláním, případně až v mikropočítači uzlu. Zadavatel tedy nepotřebuje přesně znát nastavení jednotlivých registrů řadiče SJA1000. Jednotlivá nastavení lze uložit do textového souboru a případně je i později otevřít.

Činnost ověřovacího programu pro vývojový systém s mikropočítačem C167CR spočívá v cyklickém vysílání datových zpráv daného formátu. Ovládání je provedeno

pomocí klávesnice a potenciometrů (počet kroků a rychlost) a na LCD displeji se zobrazují potřebné údaje.

Programování uzlu je provedeno ve spojové vrstvě protokolu CAN. Pro jeho lepší začlenění do průmyslových aplikací by bylo vhodné vytvořit také možnost programování v aplikační vrstvě. Zde jsou nastíněny pouze základní principy programování v aplikační vrstvě CAL a její rozšířenější verzi CANopen. Pro případné využití v automobilech, však není této možnosti tolik zapotřebí, neboť se vystačí "pouze" s vrstvou spojovou. Z tohoto důvodu by tedy záleželo na konkrétním využití tohoto uzlu s ohledem na jeho rozšíření o možnost programování v aplikační vrstvě.

Takto navržený uzel splňuje dané požadavky a jeho realizace je poměrně jednoduchá a cenově dostupná. Je tedy možno ho připojit k nějakému nadřazenému systému komunikujícímu po CAN sběrnici.

Literatura

[1] MATOUŠEK, David. *Práce s mikrokontoléry ATMEL AVR*.. Praha: BEN, 2003. ISBN 80-7300-088-1

[2] VÁŇA, Vladimír. *Mikrokontroléry ATMEL AVR – Programování v jazyce C*. Praha: BEN, 2003. ISBN 80-7300-102-0

[3] KADLEC, Václav. *Učíme se programovat v Delphi a jazyce Object Pascal*. Praha: Computer Press, 2001. ISBN 80-7226-245-9

[4] PHILIPS Semiconductors, *SJA1000 – Stand-alone CAN controller – Data Sheet*, Philips Semiconductors, 2000. http://www.semiconductors.philips.com/acrobat_download/datasheets/SJA1000_3.pdf

[5] PHILIPS Semiconductors, *PCA82C250 – CAN controller interface – Data Sheet*, Philips Semiconductors, 2000. http://coecsl.ece.uiuc.edu/ge420/datasheets/PCA82C250_5.pdf

[6] RYDLO, Pavel. *Krokové motory a jejich řízení – studijní materiály*. Liberec, 2000.

[7] VSB, *Pohony s krokovými motorky* http://www.fei.vsb.cz/kat453/www453/soubory/texty/ucebni_texty/se/cast_C_el_pohony/se_eph_c1_krokac_02_teorie.pdf

[8] RS Components. *Stepper motor driver IC SAA 1027 – Data Sheet*. RS Components, 1997. http://www.sfu.ca/phys/430/datasheets/saa1027.pdf

[9] HLAVA, Jaroslav. *Prostředky automatického řízení - skripta*. Praha, 2000.

[10] GROSMAN, Josef. *Řídící počítačové systémy – nepublikované přednášky*.

[11] EHL elektronika. *CAN - Controller Area Network*. EHL elektronika, 2001.

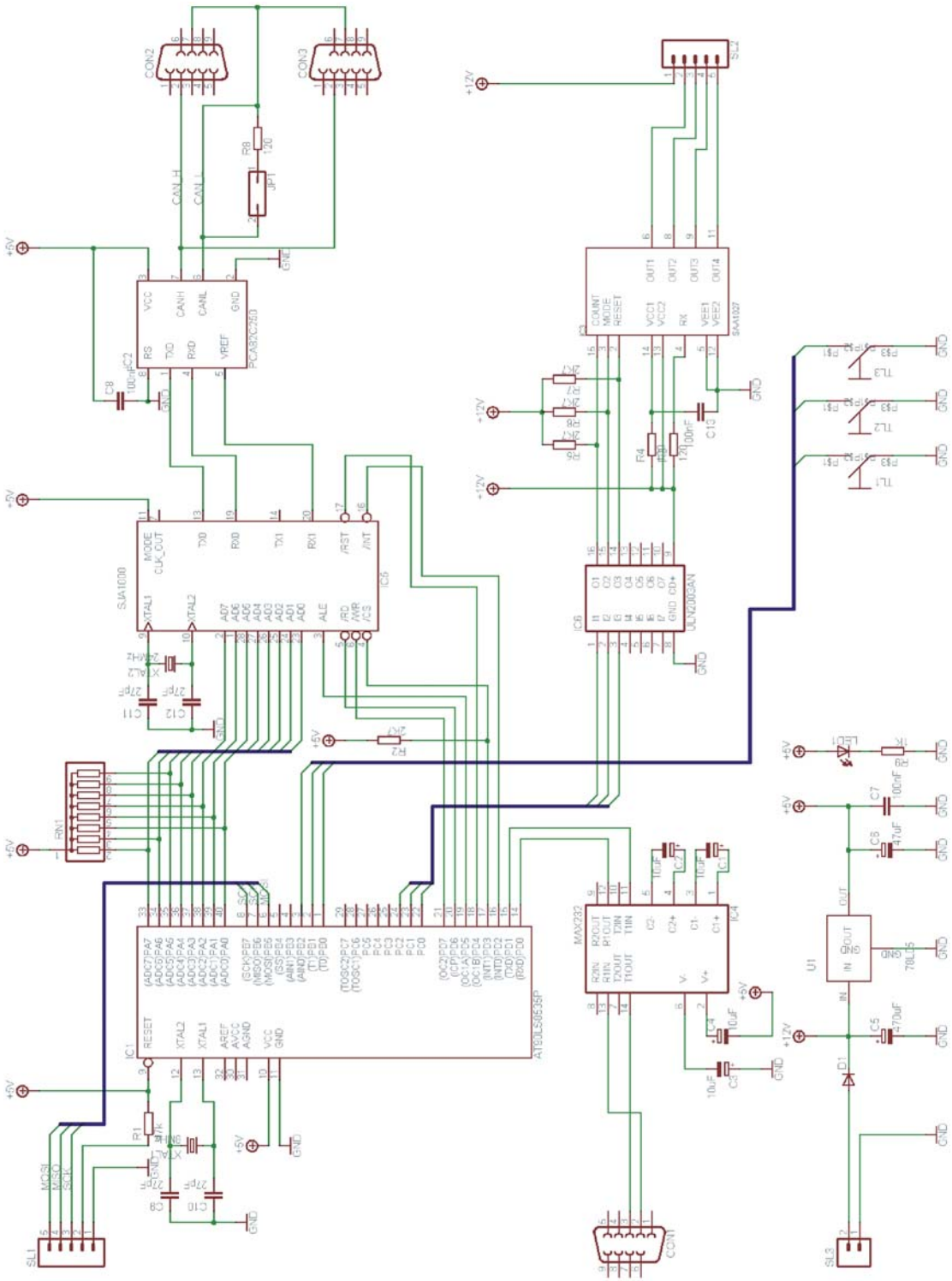
[12] CVUT, *CAN - Controller Area Network* http://fieldbus.feld.cvut.cz/can/

[13] CVUT, *Komunikační protokol CANopen* http://dce.felk.cvut.cz/drs/cviceni/can/doc/popis_canopen.pdf

[14] HITEX (UK) Ltd. *Evaluating And Training Board 167IO User Guide*, Hitex (UK) Ltd. 1997

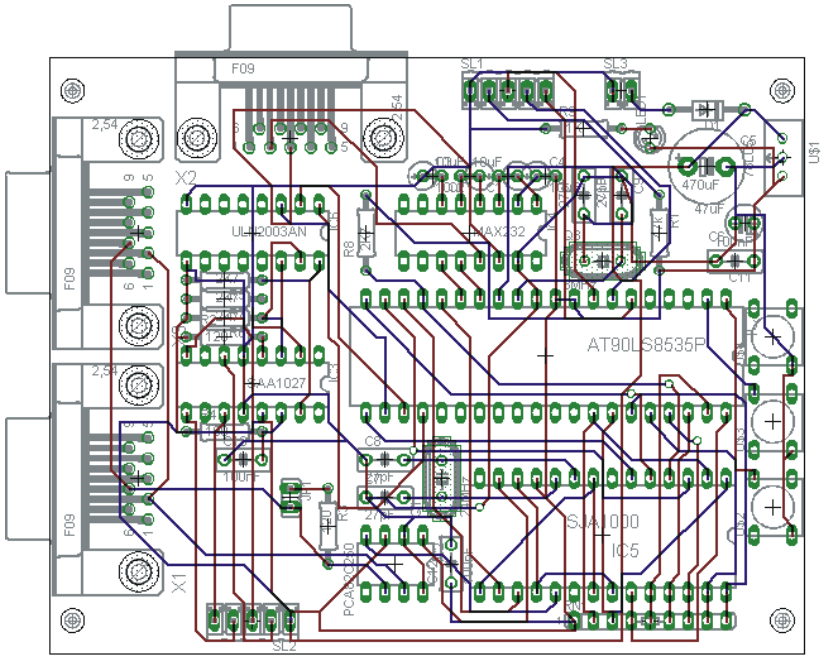
Přílohy

Příloha A: Kompletní schéma zapojení

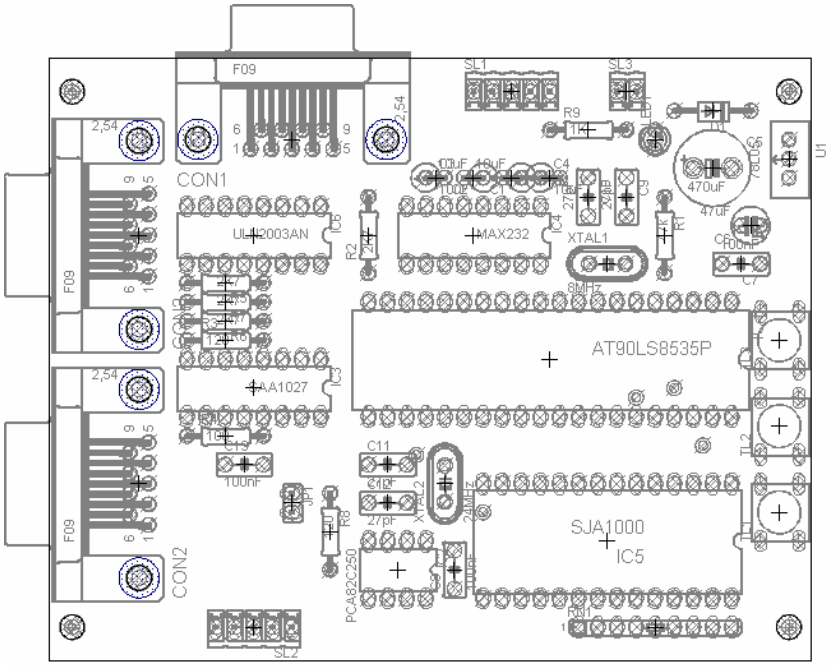


Obr. 32 Schéma zapojení

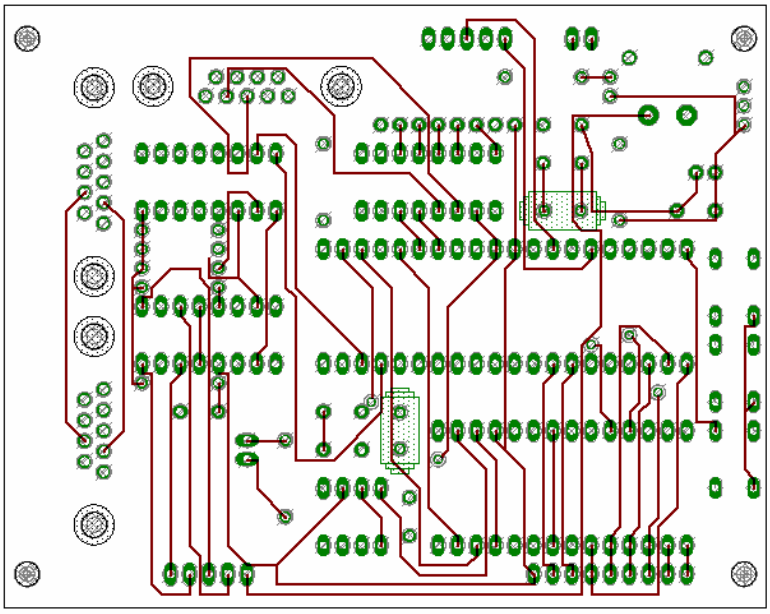
Příloha B: Návrh desky plošných spojů



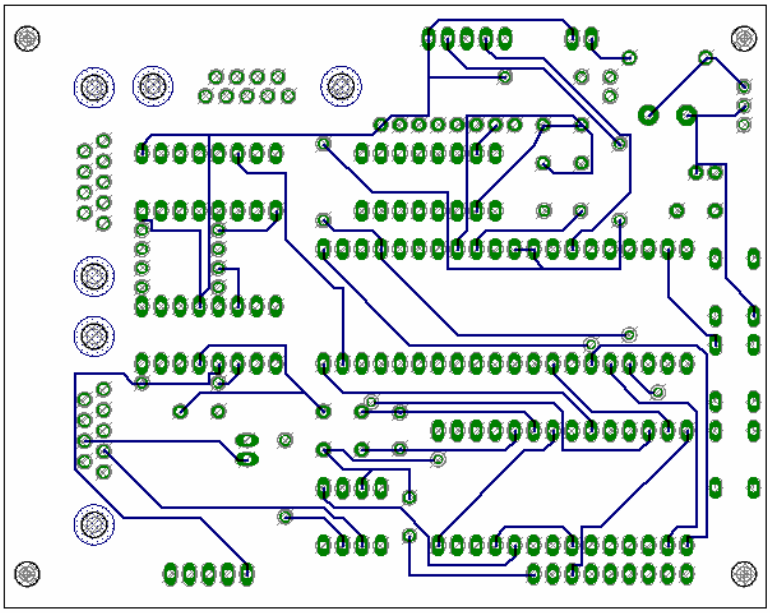
Obr. 33 Deska plošných spojů – komplet



Obr. 34 Deska plošných spojů – rozmístění součástek



Obr. 35 Deska plošných spojů – strana součástek



Obr. 36 Deska plošných spojů – strana spojů

Příloha C: Seznam použitých součástek

Položka	Počet	Reference	Hodnota
1	1	IC1	AT90S8535
2	1	IC2	PCA82C250
3	1	IC3	SAA1027
4	1	IC4	MAX232
5	1	IC5	SJA1000
6	1	IC6	ULN2003A
7	1	U1	78L05
8	1	XTAL1	8MHz
9	1	XTAL2	24MHz
10	1	RN1	3k9
11	1	R1	47k
12	4	R2, R5, R6, R7	2k7
13	2	R3, R8	120Ω
14	1	R4	100Ω
15	1	R9	1k
16	4	C1, C2, C3, C4	10μF
17	1	C5	470μF
18	1	C6	47μF
19	3	C7, C8, C13	100nF
20	4	C9, C10, C11, C12	27pF
21	1	D1	RL20F
22	1	LED1	
23	2	SL1, SL2	konektor 5
24	1	SL3	konektor 2
25	3	CON1, CON2, CON3	konektor D-SUB
26	3	TL1, TL2, TL3	tlacitko

Tab.9 Seznam použitých součástek

Příloha D: Řídící program mikrokontroléru uzlu CAN

```
/*******************************************
Project : CANMotor
Version : 1.0
Date    : 6.5.2006
Author  : Jakub Horák
Company : TUL

Chip type           : AT90S8535
Clock frequency     : 8,000000 MHz
Memory model        : Small
Internal SRAM size  : 512
External SRAM size  : 0
Data Stack size     : 128
*******************************************/

#include <90s8535.h>
#include <stdio.h>
#include <delay.h>
#include <CANinout.h>
#include <CANregs.h>

#define res PORTD.4

#define SAM       0x80
#define SJW       0xC0

//EEPROM proměnné
eeprom unsigned char eebaud    ;   //baudrate
eeprom unsigned char eeoutmode ;   //output mode
eeprom unsigned char eeclock   ;   //clock out
eeprom unsigned char eecode0   ;   //acceptance code
eeprom unsigned char eecode1   ;
eeprom unsigned char eecode2   ;
eeprom unsigned char eecode3   ;
eeprom unsigned char eemask0   ;   //acceptance mask
eeprom unsigned char eemask1   ;
eeprom unsigned char eemask2   ;
eeprom unsigned char eemask3   ;
eeprom unsigned char eeformat  ;   //frame format- standard x extended
eeprom unsigned char eeNodeIDH ;   //NodeID - standard
eeprom unsigned char eeNodeIDL ;
eeprom unsigned char eeNodeID4 ;   //NodeID - extended
eeprom unsigned char eeNodeID3 ;
eeprom unsigned char eeNodeID2 ;
eeprom unsigned char eeNodeID1 ;

//Globální proměnné
unsigned char rbuffer[20];
unsigned char sbuffer[20];
unsigned int msgread = 0;
unsigned int precteno = 0;
unsigned int format = 0;

unsigned char s='';
unsigned char znak='';
unsigned char tlacitko='';
char str[20]={''};
char str1[20]={''};
```

```
unsigned char RegH=0xFF;
unsigned char RegL=0x72;
unsigned int  Reg=0xFF72;
unsigned int  Steps=0;
unsigned char Direction=0;
unsigned int  n=0;
unsigned int  i=0;
unsigned int  ii=0;
unsigned int  j=0;

bit Konec = 0;
bit tlA=0;
bit tlB=0;
bit tlC=0;

// Příjem zprávy -----------------------------------------------------
void ReceiveMessage(void)
{
  unsigned int i;
  for (i=0; i<=12; i++)
    rbuffer[i] = inport(16+i);     //přečtení přijaté zprávy rbuffer
  outport(1, 0x04);
}

// Přerušení od vnějšího vstupu od řadiče CAN SJA1000 ----------------
interrupt [EXT_INT0] void  ext_int0(void)
{
 unsigned char intstat;
 intstat = inport(3);
 if (intstat & 0x01)
 {
   ReceiveMessage();               //příjem zprávy
   format = 0;                     //zjištění formátu přijaté zprávy

   if ((rbuffer[0] & 0x80) != 0) format = 1;
   msgread = 1;                    //nastavení příznaku přijaté zprávy

 }
}

//Přerušení při přijetí znaku po sériovém kanálu UART
interrupt [UART_RXC] void  uart_int(void)
{
  s=getchar();                      //získání znaku
  if (s == 's') znak=s;

  if (znak == '') tlacitko=s;

  if (znak == 's')                  //nastavení nových parametrů
  {
    tlacitko='';
    str[ii]=s;
    ii=ii+1;
    if (ii>=17)
      {
        ii=0;
        znak='';
        precteno=1;
      }
  }
```

```
}

//Přerušení od časovače TIMER1
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
#asm("cli")

if (Steps != 0x0000)               //Jet "Steps" kroků
  {
    PORTC.2 =! PORTC.2;
    n++;
    if (n == Steps)                      //Zastavit motorek
    {
       Konec = 1;
       n = 0;
       PORTC.0 = 1;
       TIMSK = 0x00;
    }
  }
if (Steps == 0x0000)                     //Roztočit motorek
  PORTC.2 =! PORTC.2;


TCNT1H=RegH;
TCNT1L=RegL;

#asm("clv")
#asm("sei")

}
/*****************************************************************/

// Vyslání zprávy ------------------------------------------------
void TransmiteMessage(void)
{
  unsigned int i;
  for (i=0; i<=12; i++)
    outport(16+i, sbuffer[i]);      //outport(TXBUFFER+i, sbuffer[i]);
  outport(1, 0x01);           //outport(COMMAND, TXRQ);
}

// Nastavení parametrů (baudrate,acceptance filtering) ---------------
void SetParameter(void)
{
  unsigned char BTR0, BTR1 = 0;
  unsigned char BAUDRATE = 0;

  BAUDRATE = eebaud;
  switch (BAUDRATE)
  {
    case _10kBit:
        BTR0 = 0x31 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _20kBit:
        BTR0 = 0x18 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _50kBit:
```

```
        BTR0 = 0x09 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _100kBit:
        BTR0 = 0x04 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _125kBit:
        BTR0 = 0x03 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _250kBit:
        BTR0 = 0x01 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _500kBit:
        BTR0 = 0x00 | SJW;
        BTR1 = 0x6F | SAM;
        break;

    case _800kBit:
        BTR0 = 0x00 | SJW;
        BTR1 = 0x2A;
        break;

    case _1MBit:
        BTR0 = 0xC0 | SJW;
        BTR1 = 0x18;
        break;

    default:
        BTR0 = 0x04 | SJW;
        BTR1 = 0x6F | SAM;
  }
  outport(6, BTR0);              //časování bitů
  outport(7, BTR1);

  outport(8,  eeoutmode);        //výstupní mód (normal)
  outport(31, eeclock);          //výstupní frekvence

  outport(16, eecode0);         //filtrování přijímaných zpráv po CAN
  outport(17, eecode1);
  outport(18, eecode2);
  outport(19, eecode3);
  outport(20, eemask0);
  outport(21, eemask1);
  outport(22, eemask2);
  outport(23, eemask3);

}

// inicializace přerušení -------------------------------------------
void InitInterrupt(void)
{
  GIMSK = 0x40;         //nastavení externího přerušení od INT0
  MCUCR = 0x02;         //aktivace přerušení sestupnou hranou
```

```
  outport(4, 0x01);     //přerušení po přijetí zprávy
}

// inicializace CAN řadiče ----------------------------------------
void InitCAN(void)
{
  outport(0, 0x01);    //reset mode
  outport(0, 0x09);    //acceptance filter - single mode
  SetParameter();
  InitInterrupt();
  outport(0, 0x08);    //operating mode
}

/*****************************************************************
***/
// Vyslání CAN frame standard (identifikátor 11bits + data)
void SendMsg(unsigned int NodeID, char DLC, unsigned char data[8])
{
int i;
NodeID = NodeID << 5;           //identifikátor
DLC &= 0x0F;                  //počet datových bytů

sbuffer[0] = DLC;
sbuffer[1] = NodeID / 256;
sbuffer[2] = NodeID & 0xFF;
if (DLC > 0)
  for (i=0; i<DLC; i++ )
    sbuffer[i+3] = data[i];   //naplnění registru sbuffer
TransmiteMessage();           //vyslání zprávy
}

// Vyslání CAN frame extended (identifikátor 29bits + data)
void SendMsgEXT(unsigned int NodeIDH,unsigned int NodeIDL, char DLC,
unsigned char data[8])
{
int i;
NodeIDH = ((NodeIDL >>13)&0x0007)|(NodeIDH << 3);  //identifikátor
NodeIDL = NodeIDL << 3;         //počet datových bytů
DLC &= 0x0F;

sbuffer[0] = DLC+128;
sbuffer[1] = NodeIDH / 256;
sbuffer[2] = NodeIDH & 0xFF;
sbuffer[3] = NodeIDL / 256;
sbuffer[4] = NodeIDL & 0xFF;
if (DLC > 0)
  for (i=0; i<DLC; i++ )
    sbuffer[i+5] = data[i];     //naplnění registru sbuffer
TransmiteMessage();             //vyslání zprávy
}

/*****************************************************************/

void main(void)
{
unsigned int NodeID = 0;
unsigned int NodeIDEXTH = 0;
unsigned int NodeIDEXTL = 0;
unsigned int DLC = 0;
unsigned char data[9];
```

```
UCR=0x98;         //povolení přerušení po příjmu znaku
UBRR=0x33;        //nastavení přenosové rychlosti UARTu na 9600Baud/s

DDRB.0=0;         //tlacitko 1
DDRB.1=0;         //tlacitko 2
DDRB.2=0;         //tlacitko 3
PORTB=0xFF;       //!!!!!!

DDRD=0xF8;
PORTD=0x04;       //INT0 Pull-up rezistor

DDRC=0xFF;        //portC jako výstupní (ovládání motorku)
PORTC.0=0;        //reset (!opačná logika)
PORTC.1=1;        //směr
PORTC.2=1;        //hodinové impulsy

TCNT1H=RegH;      //přednastavení časovače x=f0/(preddelicka*f)
TCNT1L=RegL;
TCCR1A=0x00;
TCCR1B=0x04;      //předdělička 256

res = 1;    //reset puls pro SJA1000
res = 0;
res = 1;

  format = eeformat;                      //nastavení identifikátoru
  if (format == 0x00)                     //standardní formát
   {
    NodeID = eeNodeIDH*256+eeNodeIDL;
   }

  if (format == 0xFF)                     //rozšířený formát
   {
    NodeIDEXTH = eeNodeID4*256+eeNodeID3;
    NodeIDEXTL = eeNodeID2*256+eeNodeID1;
   }


InitCAN();                          //inicializace CAN z EEPROM

#asm("sei")

while (1)
 {
   if (precteno == 1)               //načtení hodnot nového nastavení do
EEPROM
   {
     precteno=0;

     for (i=0;i<=16;i++)
      {
         putchar(str[i]);
      }

     eebaud    = str[1];
     eeoutmode = str[2];
     eeclock   = str[3];
     eecode0   = str[4];
     eecode1   = str[5];
     eecode2   = str[6];
     eecode3   = str[7];
```

```
  eemask0   = str[8];
  eemask1   = str[9];
  eemask2   = str[10];
  eemask3   = str[11];
  eeformat  = str[12];

  format = eeformat;
  if (format == 0x00)
   {
    eeNodeIDH = str[13];
    eeNodeIDL = str[14];
    NodeID = eeNodeIDH*256+eeNodeIDL;
   }

  if (format == 0xFF)
   {
    eeNodeID4 = str[13];
    eeNodeID3 = str[14];
    eeNodeID2 = str[15];
    eeNodeID1 = str[16];
    NodeIDEXTH = eeNodeID4*256+eeNodeID3;
    NodeIDEXTL = eeNodeID2*256+eeNodeID1;
   }

  InitCAN();         //inicializace CAN z EEPROM s novými parametry
}

if (tlacitko == 'c')          //vyslání hodnot aktuálního nastavení
 {
   tlacitko = '';
   str1[0]  ='c';
   str1[1]  = eebaud;
   str1[2]  = eeoutmode;
   str1[3]  = eeclock;
   str1[4]  = eecode0;
   str1[5]  = eecode1;
   str1[6]  = eecode2;
   str1[7]  = eecode3;
   str1[8]  = eemask0;
   str1[9]  = eemask1;
   str1[10] = eemask2;
   str1[11] = eemask3;
   str1[12] = eeformat;

   format = eeformat;
   if (format == 0x00)
    {
     str1[13] = eeNodeIDH;
     str1[14] = eeNodeIDL;
     str1[15] = 0x00;
     str1[16] = 0x00;
    }

   if (format == 0xFF)
    {
     str1[13] = eeNodeID4 ;
     str1[14] = eeNodeID3;
     str1[15] = eeNodeID2;
     str1[16] = eeNodeID1;
    }
```

```
      for (i=0;i<=16;i++)
      {
         putchar(str1[i]);
      }
    }

   if (Konec == 1)              //vyslání 0xFFFF na CAN po ujetí kroků
    {
      DLC = 2;
      data[0] = 0xFF;
      data[1] = 0xFF;
      if (format == 0xFF)
         SendMsgEXT(NodeIDEXTH, NodeIDEXTL, DLC, data); //rozšířený
      else
         SendMsg(NodeID, DLC, data);                    //standardní
      Konec = 0;
    }
   if (msgread == 1)                //vyhodnocení přijaté zprávy
     {
      msgread = 0;

      //Formát datového rámce:
      //Směr: 1 byte   ;   Pocet kroků: 2 byte   ;   Rychlost: 2 byte
      j=0;
      if (format == 0xFF) j=2;

      Reg = ((rbuffer[6+j]*256+rbuffer[7+j]))/4 + 100;   //rychlost
otáčení
      RegH = 0xFF - Reg / 256;
      RegL = 0xFF - Reg & 0xFF;

      Steps = ((rbuffer[4+j]*256+rbuffer[5+j]));   //počet kroků
      Direction = rbuffer[3+j];                    //směr

      if (Direction == 0x00)    //STOP
        {
          PORTC.0 = 1;
          TIMSK = 0x00;         //zastavit časovač
        }
      if (Direction == 0x0F)    //VPRAVO
        {
          PORTC.0 = 0;
          PORTC.1 = 0;           //směr ->
          TIMSK = 0x04;         //spustit časovač
        }
      if (Direction == 0xF0)    //VLEVO
        {
          PORTC.0 = 0;
          PORTC.1 = 1;          //směr <-
          TIMSK = 0x04;         //spustit časovač
        }
     }

   if (((PINB.0 == 0)|(tlacitko == 'r'))&(tlA == 0))    //vpravo
      {
        tlA=1; tlB=0; tlC=0;

        PORTC.0=0;
       PORTC.1=0;      //směr ->
       TIMSK=0x04;
      }
```

```
   if (((PINB.1 == 0)|(tlacitko == 'l'))&(tlB == 0))   //vlevo
      {
        tlA=0; tlB=1; tlC=0;

        PORTC.0=0;
       PORTC.1=1;       //směr <-
       TIMSK=0x04;
      }
   if (((PINB.2 == 0)|(tlacitko == 'z'))&(tlC == 0))   //stop
      {
        tlA=0; tlB=0; tlC=1;

        PORTC.0=1;       //vyresetování motorku
        TIMSK=0x00;
      }
 };
}
```

Příloha E: Clock Divider Registr CDR

7	6	5	4	3	2	1	0
CAN mode	CBP	RXINTEN	0	clock off	CD.2	CD.1	CD.0

Obr. 37 Clock Divider Registr (CDR) - adresa 31

CD.2	CD.1	CD.0	výstupní frekvence
0	0	0	$f_{osc}/2$
0	0	1	$f_{osc}/4$
0	1	0	$f_{osc}/6$
0	1	1	$f_{osc}/8$
1	0	0	$f_{osc}/10$
1	0	1	$f_{osc}/12$
1	1	0	$f_{osc}/14$
1	1	1	f_{osc}

Tab. 10 Nastavení hodnoty výstupní frekvence na vývodu CLKOUT řadiče SJA1000

Příloha F: Mapa adres řadiče SJA1000 – PeliCAN mode

CAN ADDRESS	OPERATING MODE				RESET MODE	
	READ		WRITE		READ	WRITE
0	mode		mode		mode	mode
1	(00H)		command		(00H)	command
2	status				status	
3	interrupt				interrupt	
4	interrupt enable		interrupt enable		interrupt enable	interrupt enable
5	reserved (00H)				reserved (00H)	
6	bus timing 0				bus timing 0	bus timing 0
7	bus timing 1				bus timing 1	bus timing 1
8	output control				output control	output control
9	test		test		test	test
10	reseved (00H)				reseved (00H)	
11	arbitration lost capture				arbitration lost capture	
12	error code capture				error code capture	
13	error warning limit				error warning limit	error warning limit
14	RX error counter				RX error counter	RX error counter
15	TX error counter				TX error counter	TX error counter
16	RX frame information SFF	RX frame information EFF	TX frame information SFF	TX frame information EFF	acceptance code 0	acceptance code 0
17	RX identifier 1	RX identifier 1	TX identifier 1	TX identifier 1	acceptance code 1	acceptance code 1
18	RX identifier 2	RX identifier 2	TX identifier 2	TX identifier 2	acceptance code 2	acceptance code 2
19	RX data 1	RX identifier 3	TX data 1	TX identifier 3	acceptance code 3	acceptance code 3
20	RX data 2	RX identifier 4	TX data 2	TX identifier 4	acceptance mask 0	acceptance mask 0
21	RX data 3	RX data 1	TX data 3	TX data 1	acceptance mask 1	acceptance mask 1
22	RX data 4	RX data 2	TX data 4	TX data 2	acceptance mask 2	acceptance mask 2
23	RX data 5	RX data 3	TX data 5	TX data 3	acceptance mask 3	acceptance mask 3
24	RX data 6	RX data 4	TX data 6	TX data 4	reseved (00H)	
25	RX data 7	RX data 5	TX data 7	TX data 5	reseved (00H)	
26	RX data 8	RX data 6	TX data 8	TX data 6	reseved (00H)	

CAN ADDRESS	OPERATING MODE				RESET MODE	
	READ		WRITE		READ	WRITE
27	(FIFO RAM)	RX data 7		TX data 7	reserved (00H)	
28	(FIFO RAM)	RX data 8		TX data 8	reserved (00H)	
29	RX message counter				RX message counter	
30	RX buffer start address				RX buffer start address	RX buffer start address
31	clock divider		clock divider		clock divider	clock divider
32	internal RAM address 0 (FIFO)				internal RAM address 0	internal RAM address 0
33	internal RAM address 0 (FIFO)				internal RAM address 1	internal RAM address 1
|	|		|		|	|
95	internal RAM address 63 (FIFO)				internal RAM address 63	internal RAM address 63
96	internal RAM address 64 (TX buffer)				internal RAM address 64	internal RAM address 64
|	|		|		|	|
108	internal RAM address 76 (TX buffer)				internal RAM address 76	internal RAM address 76
109	internal RAM address 77 (free)				internal RAM address 77	internal RAM address 77
110	internal RAM address 78 (free)				internal RAM address 78	internal RAM address 78
111	internal RAM address 79 (free)				internal RAM address 79	internal RAM address 79
112	(00H)				(00H)	
|	|		|		|	|
127	(00H)				(00H)	

Obr. 38 Mapa adres řadiče SJA1000 – PeliCAN mode viz.[4]